Theoretical Computer Science 642 (2016) 1–23

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Deadness and how to disprove liveness in hybrid dynamical systems

Eva M. Navarro-López *, Rebekah Carter

School of Computer Science, The University of Manchester, Oxford Road, Kilburn Building, Manchester, M13 9PL, UK

ARTICLE INFO

Article history:
Received 13 January 2015
Received in revised form 15 March 2016
Accepted 10 June 2016
Available online 16 June 2016
Communicated by P. Aziz Abdulla

Keywords:
Hybrid systems
Liveness
Stability analysis
Discontinuous systems
Hybrid automata

ABSTRACT

What if we designed a tool to automatically prove the dynamical properties of systems for which analytic proof is difficult or impossible to obtain? Such a tool would represent a significant advance in the understanding of complex dynamical systems with nonlinearities. This is precisely what this paper offers: a solution to the problem of automatically proving some dynamic stability properties of complex systems with multiple discontinuities and modes of operation modelled as hybrid dynamical systems. For this purpose, we propose a reinterpretation of some stability properties from a computational viewpoint, chiefly by using the computer science concepts of safety and liveness. However, these concepts need to be redefined within the framework of hybrid dynamical systems. In computer science terms, here, we consider the problem of automatically disproving the liveness properties of nonlinear hybrid dynamical systems. For this purpose, we define a new property, which we call *deadness*. This is a dynamically-aware property of a hybrid system which, if true, disproves the liveness property *by means of a finite execution*. We formally define this property, and give an algorithm which can derive deadness properties automatically for a type of liveness property called inevitability. We show how this algorithm works for three different examples that represent three classes of hybrid systems with complex behaviours.

1. Introduction

A hybrid dynamical system is a mathematical model for a part of the real world where discrete and continuous parts interact with each other. Such systems can model all kinds of situations, from biological systems [21,31] to a controller interacting with its environment [67,70], from electronic circuits [38,53] to mechanical systems [51].

Suppose we have a hybrid system, which is modelled in a particular framework. We might try analysing this model mathematically to work out what would happen in the system, using Lyapunov stability theory [15,23,33,48,54]. However, it is typically unclear what the behaviour patterns of a general hybrid system are, due to the interactions of the continuous dynamics and the discrete transitions. So although we can define mathematically what we would like to be true, it is not necessarily possible to decide whether this is actually true.

As a second choice, maybe we consider simulation as a method for discovering what the behaviour patterns of the system are. With this method we can get a very clear idea of what might happen for particular start states, but it is hard to know that we have covered all possible behaviour patterns of a system, no matter how many simulations we run.

* Corresponding author. Tel.: +44 161 27 56209; fax: +44 161 27 56204.
E-mail addresses: eva.navarro@manchester.ac.uk (E.M. Navarro-López), racarter174@gmail.com (R. Carter).

http://dx.doi.org/10.1016/j.tcs.2016.06.009

This leads to a third option of analysing the system: formal verification. This involves analysing the hybrid system using automatic methods to see if it conforms to some desired pattern. The verification process involves specifying the model of the system, specifying the property we wish it to conform to, and then checking if the model satisfies the desired property. This checking phase can be achieved with either logical deduction (deductive verification or theorem proving) [2,58], or an exhaustive search of the states in the system [19,63].

Whichever method is used to find the relationship between the system and the property, we must first have a specification of the system itself, which can be framed in many terms; we use the *hybrid automaton* framework [4,42,51]. With this background model, we then specify the patterns we wish the system to obey, which are usually framed as *logical properties* in some defined logic, although other approaches have been proposed [17]. Some possible logics include linear temporal logic (LTL) [59], computation tree logic (CTL) [27], and the class of various metric temporal logics [5,44].

In hybrid dynamical systems, the vast majority of logical properties studied have been *invariance properties*, which say that a system will always satisfy some logical expression for all time [20,36,63]. These invariance properties are a subset of the more general class of *safety* properties, which have an informal definition which says that a 'bad thing' never happens, and is the type of properties that have attracted more attention in the field of hybrid systems, together with model checking techniques [65]. It is interesting to highlight the application of model checking techniques to automatically generate switching controllers for linear hybrid automata [12,13]. The approach proposed in our paper has not been designed for controller synthesis. However, it could be used to evaluate the performance of closed-loop systems by automatically checking the satisfaction of some dynamical properties after a controller has been applied to the system. We stress the fact that we deal with nonlinear hybrid automata. This entails more difficulties than for linear hybrid automata for the automatic generation of controllers. Verification techniques for nonlinear systems are not as common as for linear systems. A recent tool for the verification of properties (model validation and parameter synthesis) for hybrid systems with nonlinear dynamics – in particular, piecewise nonlinear systems – is the Breach tool [25]. We also highlight the verification for nonlinear hybrid automata using barrier certificates [62].

The complementary property to safety is *liveness*, which has been barely touched as a formal verification problem for continuous-time systems. Some kind of liveness is implicit in various dynamical properties such as strictly decreasing Lyapunov functions, periodic orbits, and proving that solutions of differential equations exist, however explicit statement and computational proof of liveness properties has only happened in certain classes of hybrid systems. These classes include linear hybrid automata [6], where the dynamics are always solvable, and also systems for which piecewise constant bounds on the derivatives can be given [39], where approximations of the reachable space are easy to find. There is also recent work on proving sub-classes of liveness in continuous and hybrid systems: inevitability properties [18,26,66] and region stability [49,60,61]. Inevitability says that 'eventually a certain region of the state-space is reached', and is the simplest of the liveness properties both for specification and proof. Liveness has also been considered in examples, for instance [43] looks at safety and liveness for a controller for an automatic intersection, and [28] proves liveness properties showing that robots move in a certain way.

There are also academics who consider time-bounded liveness properties instead of liveness properties on infinite time, and these include [11,32]. Such bounded liveness properties are effectively safety properties in the way they are proved, so are generally considered less complex. However, there are some essential properties of systems which cannot be transformed into a time-bounded form, like infinite recurrence of states for instance. We should also distinguish liveness from the similarly-named concept of livelock which comes from the theory of parallel processes. Livelock and deadlock are both notions of two (or more) processes interfering with each other to stop useful motion occurring. Deadlock is where the two systems block each other from any further motion, whereas livelock is when the states of the systems keep changing, but the desired thing never happens. In this sense, both deadlock and livelock are like counter-examples to liveness, as they prove that the desired thing does not happen. In the world of parallel programs, tight definitions of deadlock and livelock are given by [69], and in hybrid control systems definitions are given by [1].

In this paper, we consider (unbounded) safety and liveness properties that can be written in a logic such as linear temporal logic (LTL), and will give a formal definition for these properties on hybrid automata, using the ideas of [3] who defined what a safety or liveness property is in concurrent programs. In dynamical systems, liveness properties can characterise many useful properties, as liveness says that something eventually becomes true. For instance, liveness can characterise the idea of convergence to some desired state, a key part of Lyapunov stability theory. The property of always returning to some desired state can also be characterised by a liveness property. In this paper we relate liveness properties to some stability-related properties of hybrid dynamical systems, mainly, global attractivity. Attractivity encompasses the idea that a system trajectory will keep getting closer and closer to a desired point in space. Stability and attractivity capture the intuition that we have about good behaviour of a dynamical system, but are not easy to prove automatically. We highlight that the stability-like properties covered by our approach are conceptually different than practical stability. Practical stability was originally proposed to deal with practical control problems, mainly, the study of finite time control and stability [34, 73,74]. It is related to convergence to behaviours with pre-specified bounds during a fixed time interval. It is similar to uniform boundedness, but in contrast to uniform boundedness, in practical stability the bounds in the state space and time are pre-defined [45].

Liveness properties are characterised by the idea that, at any finite point in an execution, they could always be satisfied at some point in the future, or alternatively that the only type of execution which can disprove such a property is one of infinite length. In continuous space–time it can be very difficult to be able to find infinite length paths. If we could gain

some knowledge about the future of a finite execution then we may be able to disprove the liveness property without actually having to find the whole infinite execution. Our idea is that we can stop an execution if we know it will never satisfy the liveness property, due to some dynamical knowledge about the future of this execution. With this in mind, we define a new type of dynamically-aware property which can disprove a liveness property with a finite length path. This is the concept of *deadness*, which captures the idea that there could exist another property which, if it is true, implies that the liveness property can never hold in the system: 'if a system is dead it can never be live again'.

After defining deadness formally (Section 5), we introduce an algorithm to find deadness properties automatically on nonlinear hybrid automata (Section 6), which is a step forward in disproving liveness in hybrid systems. The algorithm works for special cases of liveness and deadness, specifically, it finds a deadness property for an inevitability property. The novel idea of our algorithm is that the verification procedure is guided by stability-like properties of the equilibria present in the system. For this, the notion of hybrid-space equilibria is introduced. In Section 7, we then discuss how to implement this algorithm and how to use it to prove deadness (therefore disproving liveness), and describe our implementation. The algorithm uses invariant sets around undesired equilibria as 'dead sets' which disprove liveness if they can be proven to be reached. To practically prove deadness, we need to find at least one execution of the hybrid automaton which eventually reaches one of the dead sets. We interpret this problem as a satisfiability (SAT) problem, and we have made a prototype implementation using MATLAB and iSAT (a hybrid system SAT solver) [30]. We highlight that the effectiveness of our prototype implementation is dependent on the available mathematical tools for finding locally stable equilibrium points in nonlinear dynamical systems.

The next section (Section 2) shows the problem addressed by this work in more detail, by means of a motivating example of a simplified model of an oilwell drillstring. This system represents a family of discontinuous systems with discontinuous state derivatives and sliding motions, which includes a wide range of systems studied in the area of discontinuous control systems and sliding-mode control. The general hybrid automaton that models these systems was proposed in [51] as the discontinuous dynamical systems (DDS) hybrid automaton. In Section 8, we will return to the motivating example to demonstrate the properties and methods presented. Moreover, we will show how our algorithm of finding deadness properties is applicable to two more examples: 1) a system that exhibits chaotic behaviour, particularly, the Lorenz system in its discontinuous version, and 2) a nonlinear hybrid automaton that has virtual equilibrium points in the discrete locations. A virtual equilibrium for a discrete location q_i is an equilibrium point for the dynamical system within q_i that does not belong to the domain of q_i, but belongs to the domain of some other location of the hybrid automaton. The concept of virtual equilibria of discrete locations in hybrid automata was first introduced in [54]. Although virtual equilibria are common in hybrid automata and their existence may affect the dynamical behaviour of hybrid automata, they have not been considered in the hybrid systems literature. In the three examples considered in this paper, we analyse a particular type of liveness property which fits in our general framework. Specifically, we will disprove the global attractivity of one of the equilibrium points of the examples.

In brief, the contribution of this paper is four-fold. First, we propose a novel framework to disprove liveness properties on general hybrid systems. In this context, we offer a new formal definition of liveness for general hybrid automata, based on the ideas of [3] and using future unaware executions to make the concept of liveness independent of the actual dynamics of the hybrid automaton, making our results applicable to more general hybrid automata than in already-published works. Second, the systems under study are a class of nonlinear hybrid automata, where the dynamical system within every discrete location is nonlinear (including linear dynamics). We consider hybrid automata with multiple isolated equilibrium points, for which some discrete locations/subsystems may have no equilibrium point, which is the typical situation in complex systems which consist of different interdependent and interlocking subsystems with different set points of each operation mode [54]. These types of systems have not been treated before in this context. Third, we define a dynamically-aware logical property called deadness for nonlinear hybrid automata and propose a practical method to find such deadness properties automatically. With this, we introduce a framework to help prove more instances of properties on hybrid systems, as using a deadness property can increase our ability to find counterexamples to liveness properties by only requiring finite executions of hybrid automata to be found. Finally, we propose a dynamically-driven verification method, by which we mean that the way the method works is guided by properties of the dynamics of the hybrid automata, which is a novel approach in verification. The dynamical properties used in the method are stability-like properties of the equilibria present in the system. These properties are either directly available or can be automatically derived from the description of the system.

The restrictions on the dynamics of the hybrid automata that our algorithm for finding dead sets can handle are two: 1) we consider equilibria that can trap executions of the hybrid automata in one location only (see Section 6), and 2) in order to build the dead sets, our hybrid automata must contain at least one stable (if the discrete location has linear dynamics) or locally asymptotically stable (if the discrete location has nonlinear dynamics) equilibrium that is outside the live set. This equilibrium point can belong to any of the discrete locations. With these restrictions, the family of systems that our implementation can handle is still more general than most of the already-existing results on proving properties of hybrid dynamical systems. Even though our algorithm is only applicable to systems with the mentioned restrictions, the theoretical framework proposed is valid for general hybrid systems.

We note that our algorithm for finding dead sets in order to disprove a liveness property does not need to find all the stable-like equilibria of the system, it only requires to find one stable-like equilibrium outside the live set.

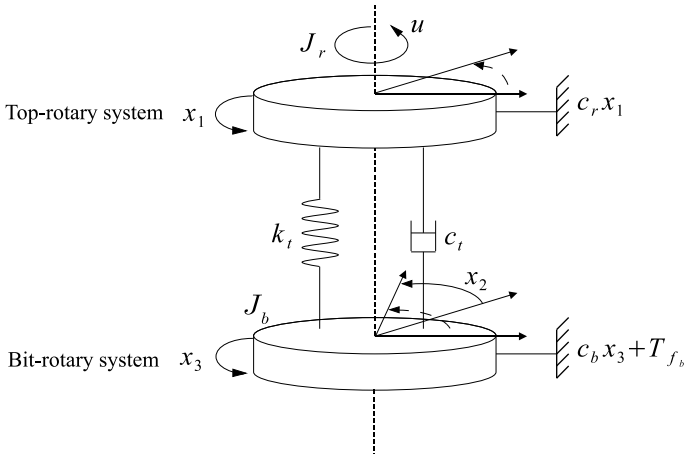

Fig. 1. Mechanical model describing the torsional behaviour of a simplified drillstring. The curved dashed arrows indicate the angular displacement of the top-rotary system and bit, and x_2 is the difference between them.

2. A motivating example

To motivate the definitions in this paper, the following discontinuous system, which can be modelled as a hybrid automaton, is considered. This system is an example of the general class of DDS hybrid automata defined in [51]. It is a model describing the torsional behaviour of a simplified oilwell vertical drillstring that exhibits multiple equilibria and periodic oscillations [51]:

$$\begin{aligned}
\dot{x}_1 &= \frac{1}{J_\mathrm{r}}\left[-(c_\mathrm{t}+c_\mathrm{r})x_1 - k_\mathrm{t}x_2 + c_\mathrm{t}x_3 + u\right],\\
\dot{x}_2 &= x_1 - x_3,\\
\dot{x}_3 &= \frac{1}{J_\mathrm{b}}\left[c_\mathrm{t}x_1 + k_\mathrm{t}x_2 - (c_\mathrm{t}+c_\mathrm{b})x_3 - T_{f_\mathrm{b}}(x_3)\right].
\end{aligned} \tag{1}$$

The dot denotes derivative with respect to time. Here x_1 and x_3 are the angular velocities of the top-rotary system and the bit, respectively, and x_2 is the difference between the two angular displacements (see Fig. 1). We combine these variables into a state vector $x = (x_1, x_2, x_3)^\mathrm{T} \in \mathbb{R}^3$. The input torque $u > 0$ and the weight on the bit ($W_\mathrm{ob} > 0$) are two varying parameters. The discontinuous friction torque is $T_{f_\mathrm{b}}(x_3) = f_\mathrm{b}(x_3)\mathrm{sign}(x_3)$, where

$$f_\mathrm{b}(x_3) = W_\mathrm{ob}R_\mathrm{b}\left[\mu_{c_\mathrm{b}} + (\mu_{s_\mathrm{b}} - \mu_{c_\mathrm{b}})\exp^{-\frac{\gamma_\mathrm{b}}{v_\mathrm{f}}|x_3|}\right]. \tag{2}$$

Here $R_\mathrm{b} > 0$ is the bit radius; μ_{s_b}, $\mu_{c_\mathrm{b}} \in (0,1)$ are the static and Coulomb friction coefficients associated with the bit; and $0 < \gamma_\mathrm{b} < 1$ and $v_\mathrm{f} > 0$ are constants. The Coulomb and static friction torque are T_{c_b} and T_{s_b}, respectively, with $T_{c_\mathrm{b}} = W_\mathrm{ob}R_\mathrm{b}\mu_{c_\mathrm{b}}$, $T_{s_\mathrm{b}} = W_\mathrm{ob}R_\mathrm{b}\mu_{s_\mathrm{b}}$. The sign function in $T_{f_\mathrm{b}}(x_3)$ is considered as:

$$\begin{aligned}
&\mathrm{sign}(x_3) = x_3/|x_3| \quad \text{if } x_3 \neq 0,\\
&\mathrm{sign}(x_3) \in [-1,1] \quad \text{if } x_3 = 0.
\end{aligned}$$

The uncertainty of the system behaviour when the velocity x_3 is zero is overcome by choosing an adequate mathematical model on the discontinuity surface $\Sigma := \{x \in \mathbb{R}^3 : x_3 = 0\}$. When $x_3 = 0$, we define T_{f_b} by Utkin's equivalent control method for sliding modes as [71]:

$$T_{f_\mathrm{b}}(x) = u_\mathrm{eq}(x) = c_\mathrm{t}x_1 + k_\mathrm{t}x_2 - (c_\mathrm{t}+c_\mathrm{b})x_3. \tag{3}$$

All the parameters and variables in our model are real numbers. Typical parameter values that we will use in this work are:

$$\begin{aligned}
c_t &= 172.3067\ \mathrm{N\,m\,s/rad}, & J_b &= 471.9698\ \mathrm{kg\,m^2}, & \mu_{c_b} &= 0.5,\\
k_t &= 861.5336\ \mathrm{N\,m/rad}, & c_r &= 425\ \mathrm{N\,m\,s/rad}, & J_r &= 2122\ \mathrm{kg\,m^2},\\
\mu_{s_b} &= 0.8, & u &= 6000\ \mathrm{N\,m}, & c_b &= 50\ \mathrm{N\,m\,s/rad},\\
R_b &= 0.155575\ \mathrm{m}, & \gamma_b &= 0.9, & v_f &= 1\ \mathrm{rad/s}.
\end{aligned}$$

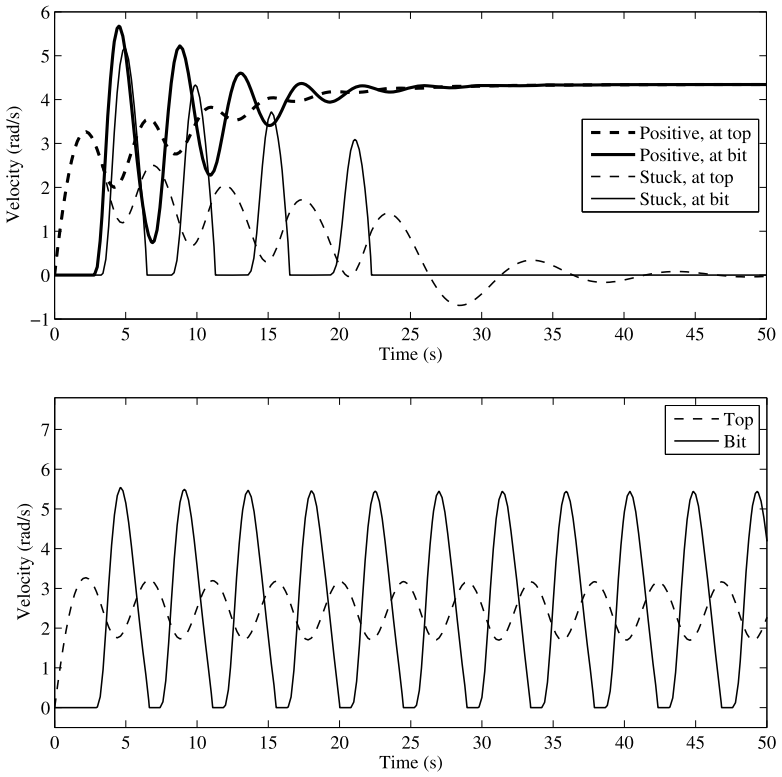

Fig. 2. Top: positive velocity and permanently stuck equilibrium behaviours. Bottom: stick-slip motion. Dashed lines represent x_1, and solid lines represent x_3. For these particular trajectories, $u = 6000$ N m with W_{ob} variable: high W_{ob} ($= 59208$ N) results in the stuck behaviour, low W_{ob} ($= 50000$ N) results in the positive velocity behaviour, and medium W_{ob} ($= 53018$ N) causes the periodic stick-slip behaviour.

This system exhibits a rich collection of behaviours depending on the competing 'strengths' of two locally attractive equilibrium points: one with $x_3 = 0$ and one with $x_3 > 0$. The values of (u, W_{ob}) vary the relative attractivity of the two equilibria, resulting in three main behaviour patterns:

- **Positive velocity equilibrium**: the bit velocity x_3, converges to a positive equilibrium value and $x_1 = x_3$.
- **Permanent stuck bit**: the bit stops rotating after some period of time and never starts again.
- **Stick-slip motion**: the bit velocity oscillates between zero and a positive velocity.

Fig. 2 shows these three behaviour patterns.

Analysing dynamical patterns of this system is very hard, as it is with many nonlinear hybrid systems. We can identify these three behaviours by simulation, but it is very difficult to know which one will actually be present in the system for any given set of parameters (see [52] for the dynamical analysis of this model). Consequently, this model of a drillstring is an ideal candidate for new methods of analysis, in particular formal verification. In Section 8, we will return to this example and show how we can disprove that the positive velocity equilibrium is globally attractive by using the permanent stuck bit equilibrium to create a deadness property. In Section 8, we will also consider two more examples: a chaotic hybrid automaton and a nonlinear hybrid automaton with virtual equilibria in the discrete locations.

3. Preliminary definitions

In this section, we introduce the key definitions which we make use of in this paper. We will firstly give those definitions relating to hybrid automata, and then we will introduce the two concepts of safety and liveness for such systems.

3.1. Hybrid automata

Hybrid automata are a useful model of hybrid dynamical systems, since they explicitly show the interaction between the continuous and the discrete parts of the system. A variation on the classical automaton idea is used to model the discrete changes in the system, with differential equations used to model the continuous motion. We do not consider inputs in this model, but the results are not limited by this restriction.

Definition 1 *([42]).* A *hybrid automaton* is a collection

$$H = (Q, E, \mathcal{X}, Dom, \mathcal{F}, Init, G, R)$$

that models a hybrid system, where

- Q is a finite set of locations.
- $E \subseteq Q \times Q$ is a finite set of edges called transitions or events.
- $\mathcal{X} \subseteq \mathbb{R}^n$ is the continuous state-space.
- $Dom : Q \to 2^{\mathcal{X}}$ is the location domain (sometimes called an invariant). It assigns a set of continuous states to each discrete location $q \in Q$, thus, $Dom(q) \subseteq \mathcal{X}$.
- $\mathcal{F} = \{f_q(x) : \ q \in Q\}$ is a finite set of vector fields describing the continuous dynamics in each location, such that $f_q : \mathcal{X} \to \mathcal{X}$. Each $f_q(x)$ is assumed to be Lipschitz continuous on the location domain for q in order to ensure that the solution exists and is unique.
- $Init \subseteq \bigcup_{q \in Q} q \times Dom(q) \subseteq Q \times \mathcal{X}$ is a set of initial states.
- $G : E \to 2^{\mathcal{X}}$ is a guard map. G assigns to each edge a set of continuous states; this set contains the states which enable the edge to be taken.
- $R : E \times \mathcal{X} \to 2^{\mathcal{X}}$ is a reset map for the continuous states for each edge. It is assumed to be non-empty, so that the dynamics can only be changed, not destroyed. ■

Definition 2 *([42]).* The *hybrid state space* is the set defined by

$$\mathcal{Z} \equiv \bigcup_{q \in Q} q \times Dom(q) \subseteq Q \times \mathcal{X}.$$

That is, the set of all pairs (q, x) which the hybrid automaton allows to exist. A *hybrid state* z is a member of the hybrid state space, or $z = (q, x) \in \mathcal{Z}$. A hybrid set W is a subset of $\mathcal{Z}$, that is $W \subseteq \mathcal{Z}$. ■

We now define how a hybrid automaton can evolve, firstly by defining the hybrid time trajectory, and secondly by defining the execution of the hybrid automaton on such a trajectory.

Definition 3 *([42]).* A *hybrid time trajectory* $\tau = \{I_i\}_{i=0}^{N}$ is a finite or infinite sequence of intervals of the real line, such that

- for all $0 \le i < N$, $I_i = [t_i, t_i']$ with $t_i \le t_i' = t_{i+1}$;
- if $N < \infty$, either $I_N = [t_N, t_N']$ with $t_N \le t_N' < \infty$, or $I_N = [t_N, t_N')$ with $t_N < t_N' \le \infty$.

The set of all hybrid time trajectories is denoted by $\mathcal{T}$. ■

For ease of notation we will use $t \in \tau$ as a shorthand for 'there is some i such that $t \in [t_i, t_i'] \in \tau$' for any $\tau \in \mathcal{T}$. We will also always write the final interval in the sequence as a closed interval $[t_N, t_N']$, but the reader may substitute $[t_N, t_N')$ if required.

When considering a finite part of an infinite time trajectory, also called a *partial* hybrid time trajectory, we will use the notation $\tau_{0,p}$, $0 \le p \le N$, $p < \infty$. This consists of the intervals $\{I_i\}_{i=0}^{p}$, where $I_i = [t_i, t_i']$ for $0 \le i < p$, and $I_p = [t_p, t_p'']$ with $t_p \le t_p'' \le t_p'$ and $t_p'' < \infty$.

We now define the execution of the system on τ. The idea is that continuous flow of the hybrid automaton occurs in every interval $[t_i, t_i']$ (when this interval is of non-zero length), and discrete transitions occur to take the end of one interval $[t_i, t_i']$ to the start of the next one $[t_{i+1}, t_{i+1}']$. This captures the behaviour of the hybrid automaton perfectly, allowing continuous flow in one location, taking us to a point when we make a discrete transition to another location, to continue continuous motion again.

Definition 4 *(Valid execution [42]).* An execution ϕ of a hybrid automaton H is a collection $\phi = (\tau, z)$ with hybrid time trajectory $\tau = \{[t_i, t_i']\}_{i=0}^{N} \in \mathcal{T}$, and $z : \tau \to \mathcal{Z}$ a product of mappings $q : \tau \to Q$ and $x : \tau \to \mathcal{X}$, satisfying:

1. Initial condition: $z(t_0) = (q(t_0), x(t_0)) \in Init$.
2. Continuous evolution: for all i such that $t_i < t_i'$, it is the case that for $t \in [t_i, t_i']$, $q(t)$ is constant and $x(t)$ is Lipschitz continuous and differentiable, $x(t) \in Dom(q(t))$, and the evolution is described by $\dot{x}(t) = f_{q(t)}(x(t))$ for $t \in [t_i, t_i')$.
3. Discrete transitions: for all $i \in \{0, 1, \ldots, N-1\}$, there is an edge $e = (q(t_i'), q(t_{i+1})) \in E$, for which $x(t_i') \in G(e)$, and $x(t_{i+1}) \in R(e, x(t_i'))$.

The set of all executions of H from the initial set $Init$ is defined by $\mathcal{E}_{H,Init}$, and the set of all executions of H with initial set equal to $\mathcal{Z}$ (the whole space) is defined by $\mathcal{E}_H$. ■

For any hybrid time trajectory τ, note that $z(t_i')$ is not necessarily equal to $z(t_{i+1})$, due to the implicit dependence of q and x on the interval of τ being considered. In general, at least the discrete state will change, so that $q(t_i') \neq q(t_{i+1})$. However, it is possible to have $q(t_i') = q(t_{i+1})$ and $x(t_i') \neq x(t_{i+1})$ as it happens in the most simple reset systems.

We now introduce the notion of a *future unaware execution*, which is a sequence of hybrid states ϕ in the hybrid state space of the hybrid automaton H, where ϕ does not have to follow the dynamics of H.

Definition 5 *(Future unaware execution).* A future unaware execution ϕ of a hybrid automaton H is a collection $\phi = (\tau, z)$ with hybrid time trajectory $\tau = \{[t_i, t_i']\}_{i=0}^{N} \in \mathcal{T}$, and $z : \tau \to 2^{\mathcal{Z}}$ is a product of multivalued mappings $q : \tau \to 2^{Q}$ and $x : \tau \to 2^{\mathcal{X}}$, satisfying:

1. Initial condition: $z(t_0) = (q(t_0), x(t_0)) \in \mathit{Init}$.
2. Continuous evolution: $x(t)$ is continuous and $x(t) \in \mathit{Dom}(q(t))$ in every interval $t \in [t_i, t_i']$ for $0 \leq i \leq N$.
3. Discrete transitions: for all $i \in \{0, 1, \ldots, N-1\}$, ϕ jumps from one location $q(t_i')$ to another $q(t_{i+1})$; this jump does not necessarily follow the guards. The continuous state x can be reset to any value in the new location domain $\mathit{Dom}(q(t_{i+1}))$.

The set of all future unaware executions of H that start from the initial set *Init* is defined by $\mathcal{E}_{U,\mathit{Init}}$, and the set with initial set $\mathcal{Z}$ (the whole space) is denoted by $\mathcal{E}_U$. ■

These future unaware executions are a generalisation of the notion of Kleene closure of the set of symbols in a finite-state automaton [40]. In both cases we know that when we consider the actual structure of the finite-state or hybrid automaton, the resulting strings/executions will exist as a subset of the symbol set closure or the future unaware set respectively. Another way of thinking about it is that the future unaware executions are all the executions possible given only a hybrid state space in which they occur.

We now define a useful order on hybrid time trajectories and hybrid automaton executions, and then use it when we classify executions into types.

- $\tau = \{I_i\}_{i=0}^{N} \in \mathcal{T}$ is a *prefix* of $\tau' = \{J_i\}_{i=0}^{M} \in \mathcal{T}$, denoted $\tau \preceq \tau'$, if either they are identical or τ is finite with $M \geq N$, $I_i = J_i$ for all $i = 0, \ldots, N-1$ and $I_N \subseteq J_N$.
- $\phi = (\tau, z)$ is a *prefix* of $\phi' = (\tau', z')$, denoted $\phi \preceq \phi'$, if $\tau \preceq \tau'$ and $z(t) = z'(t)$ for all $t \in \tau$.
- ϕ is a *strict prefix* of ϕ', denoted $\phi \prec \phi'$, if $\phi \preceq \phi'$ and $\phi \neq \phi'$.

Definition 6 *([42]).* Taking $0 \leq N \leq \infty$, we define an execution $\phi = (\tau, z)$ to be:

- *finite* if τ is a finite sequence ending with a finite interval, that is $\tau = \tau_{0,N}$ with $t_N' < \infty$ and $N < \infty$,
- *infinite* if τ is a finite sequence ending with an infinite interval *or* an infinite sequence, that is $\tau = \tau_{0,N}$ with $t_N' = \infty$, or $N = \infty$,
- *maximal* if $\nexists \phi'$ with $\phi \prec \phi'$. ■

We will assume in this work that the hybrid automaton is non-blocking (see [42]), so that maximal executions of the system are always infinite. The assumption of a non-blocking automaton simply rules out behaviours for which the model 'gets stuck' after a finite length of time. In general these will be artefacts of the mathematical model and not realistic behaviours. This is because a real-world system does not stop after a finite length of time, instead time continues and forces the system to keep evolving in some way. Hence, except in very specific cases, an infinite execution will always occur, so it is realistic for the model to be non-blocking.

We now classify some properties of the executions. Firstly notice that the set of valid executions of H is a subset of the class of future unaware executions, or $\mathcal{E}_H \subseteq \mathcal{E}_U$. The set of all executions with particular initial condition $z(t_0) \in \mathcal{Z}$ is denoted by $\mathcal{E}_{H,z(t_0)}$ for valid executions of H, and by $\mathcal{E}_{U,z(t_0)}$ for the set of future unaware executions. The valid executions of H from a set of initial conditions *Init* is denoted by $\mathcal{E}_{H,\mathit{Init}}$, and similarly $\mathcal{E}_{U,\mathit{Init}}$ for future unaware. We also define $\mathcal{E}_H^F$ and $\mathcal{E}_H^\infty$ as, respectively, the sets of all finite and infinite executions of H (similarly $\mathcal{E}_U^F$ and $\mathcal{E}_U^\infty$ for future unaware executions). These can also be combined: the set of finite valid executions which start from $z(t_0) \in \mathcal{Z}$ is $\mathcal{E}_{H,z(t_0)}^F$ for example.

3.2. Safety and liveness

In this section we will define the concepts of safety and liveness, which are descriptors for logical properties. In this work we do not use any particular logic, but it must be a future-time temporal logic which can express both safety and liveness properties on hybrid systems, for example linear temporal logic (LTL) [59] or computation tree logic (CTL) [27]. We assume that we have syntax and semantics[1] defined for the logic we are using, and define the satisfaction relation by the following.

[1] Syntax is the symbols we can use and the way these symbols can be combined, and semantics is the meaning of these symbols in the logic.

Definition 7 *(Satisfaction by infinite executions).* Consider the hybrid automaton H, and the formula φ defined in some temporal logic on H. An infinite (possibly future unaware) execution of the hybrid automaton $\phi = (\tau, z) \in \mathcal{E}_U^\infty$ is said to satisfy φ and is denoted by $\phi \vDash \varphi$, if and only if $z(t)$ satisfies the semantics of the formula φ in the logic. ■

This definition only defines how infinite executions can satisfy a property. It is necessary for the definition of deadness that we know how finite executions satisfy logical properties, so we define *chattering semantics* for expressions of the form $\phi_{0,p} \vDash \varphi$, where the last value of the execution, $z(t_p'') = (q(t_p''), x(t_p''))$, is repeated for the rest of time. This is a sensible semantics for continuous-time properties, where there is no notion of a 'next state' in the execution.

Definition 8 *(Satisfaction by finite executions).* Given a finite execution $\phi_{0,p}$, we define the *chattering extension* to this execution by $\phi_{0,p}^c = (\tau_c, z_c) \in \mathcal{E}_U^\infty$ where $\tau_c = \{[t_0, t_0'], \ldots, [t_{p-1}, t_{p-1}'], [t_p, \infty)\}$, and $z_c(t) = z_{0,p}(t)$ for $t \in \tau_{0,p}$, and $z_c(t) = z(t_p'')$ for $t \in [t_p'', \infty)$. Then, a finite execution can satisfy a formula φ by considering the equivalence

$$(\phi_{0,p} \vDash \varphi) \equiv (\phi_{0,p}^c \vDash \varphi). \quad \blacksquare \tag{4}$$

Note that this execution extended by chattering is not necessarily a valid execution of the hybrid automaton H, but will definitely belong to the set of future unaware executions. Since this is only a convention for satisfaction of logical formulae we do not worry about the real-world meaning.

We will now formally define a safety property for hybrid systems, where intuitively the idea is that 'nothing bad ever happens'. The formal definition is based on the fact that if an infinite execution is not safe, then there must have been a point in time at which the "bad thing" happened. Safety was originally defined for discrete-time systems by Lamport in [57], although we use the definition by Alpern and Schneider [3]. Note that our generalisation of this definition to hybrid systems uses future unaware executions, as whether a logical property is classified as safety is independent on the dynamics of the hybrid automaton – it only requires a hybrid state space to define the property on.

Definition 9. A formula S defined on the hybrid state space $\mathcal{Z}$ is a *safety property* iff for all future unaware infinite executions that do not satisfy S a finite prefix can be found for which all infinite extensions do not satisfy S, or more formally

$$\forall \phi \in \mathcal{E}_U^\infty \left(\phi \nvDash S \Rightarrow \exists \phi_{0,p} \in \mathcal{E}_U^F \; \phi_{0,p} \prec \phi \quad \forall \phi' \in \mathcal{E}_U^\infty (\phi_{0,p} \prec \phi' \Rightarrow \phi' \nvDash S)\right). \quad \blacksquare \tag{5}$$

Safety properties in hybrid systems have typically been reduced to invariance properties, which say that 'some bad region is never reached'. In the drillstring example, for instance, a safety property could be that we never want to get negative velocity on the drill bit (so that it is always drilling forwards into the ground).

Liveness is a property which says that 'something good eventually happens'. It has been considered before in discrete systems, mostly in the context of verification of such systems [9,14,56], but has not been formally considered in hybrid dynamical systems. Here we define liveness in the context of hybrid automata, using the ideas of Alpern and Schneider [3]. The formal definition uses the fact that if 'something good eventually happens' and at some finite point in an execution it has not already happened, then it must still be possible to satisfy the liveness property at some point in the future. We again use future unaware executions, to make the concept of liveness independent of the actual dynamics of H.

Definition 10. A formula L defined on the hybrid state space $\mathcal{Z}$ is a *liveness property* iff every finite future unaware execution of H can be extended to an infinite execution which satisfies L, or more formally

$$\forall \phi_{0,p} \in \mathcal{E}_U^F \; \exists \phi \in \mathcal{E}_U^\infty \left(\phi_{0,p} \prec \phi \wedge \phi \vDash L\right). \quad \blacksquare \tag{6}$$

The key idea of liveness properties is that they *cannot be directly disproved by a finite execution*, as at any finite time point we do not know what will happen in the future, and so the 'good thing' could still happen. In dynamical systems, the idea of liveness is most clearly related to achieving a desired goal, whether it be that of reaching a useful region of the state space or that of tending to a periodic cycle of states.

We now define the way that the definitions of safety and liveness properties translate into descriptors for executions in the hybrid automaton.

Definition 11. An infinite execution $\phi \in \mathcal{E}_U^\infty$ is called *safe* with respect to safety property S iff it satisfies the safety property, or $\phi \vDash S$. A hybrid automaton H is *safe* iff $\forall \phi \in \mathcal{E}_{H,Init}^\infty \; (\phi \vDash S)$. ■

Definition 12. An infinite execution $\phi \in \mathcal{E}_U^\infty$ is *live* with respect to liveness property L iff it satisfies the liveness property, or $\phi \vDash L$. A hybrid automaton H is *live* iff $\forall \phi \in \mathcal{E}_{H,Init}^\infty \; (\phi \vDash L)$. ■

4. Relating liveness to stability-type properties

In this section, we will look at how a typical stability-type property of hybrid dynamical systems relates to a simple liveness property. This will motivate the definition of deadness in the next section.

We will use the notation of a *ball* of radius $r > 0$ around a point $p \in \mathbb{R}^n$, defined by $B(r, p) = \{x \in \mathbb{R}^n : \|x - p\| < r\}$, where $\|\cdot\|$ denotes the 2-norm on the Euclidean space $\mathbb{R}^n$.

In a hybrid automaton, we consider an equilibrium point as a point in space at which an execution of the system, if it starts at the point, does not change its position in space as time evolves. We will denote an equilibrium point of a hybrid automaton as $\overline{z} = (\overline{q}, \overline{x}) \in \mathcal{Z}$, which will be referred to as hybrid-space equilibrium. The formal definition is given in Definition 16 in Section 6. Now, we will only consider the continuous-space equilibrium $\overline{x} \in \mathcal{X} \subseteq \mathbb{R}^n$.

Let us consider the property of attractivity on hybrid systems, and relate it to a liveness property. Attractivity says that every trajectory which starts within a certain range of an equilibrium point will tend towards that equilibrium, reaching it eventually (in possibly infinite time or an infinite number of discrete transitions). Here we just consider global attractivity. The formal definition for global attractivity in hybrid automata is

$$\overline{x} \text{ is globally attractive} \Leftrightarrow \forall \phi = (\tau, z) \in \mathcal{E}^{\infty}_{H,Init},\ \lim_{t \to t_\infty} x(t) = \overline{x},$$

with $t_\infty = \sum_i (t'_i - t_i)$, the final time in the execution ϕ.

In continuous-time systems without discontinuities, global attractivity is considered for all initial conditions in the state space, and theoretically this is how the definition should be considered in hybrid automata. However, due to the complexity of hybrid systems we would not usually expect every execution starting from every point in the system to converge to one equilibrium [54], so global attractivity of an equilibrium is not very useful with $Init = \mathcal{Z}$. This is why we have included the initial set *Init* in the allowed executions for global attractivity, as we are more likely to be interested in whether the equilibrium is attractive given a certain set of likely initial conditions, $Init \subseteq \mathcal{Z}$. This still allows for the case when $Init = \mathcal{Z}$ should we require it.

Rewriting the limit function with its classical definition gives us

$$\overline{x} \text{ is globally attractive} \Leftrightarrow \left[\forall \phi = (\tau, z) \in \mathcal{E}^{\infty}_{H,Init}, \left(\forall \epsilon, \exists \delta > 0 \left[t \in B(\delta, t_\infty) \Rightarrow x(t) \in B(\epsilon, \overline{x})\right]\right)\right]. \tag{7}$$

That is, the equilibrium $\overline{x}$ of a hybrid automaton is globally attractive if for any execution, we can select any small ball around $\overline{x}$ and guarantee if we go far enough in time that we will enter this selected ball.

This attractivity property does not lend itself to computational proof very easily, as it involves two different quantifiers intricately linked. However, we can consider a weaker condition, an inevitability property:

$$\forall \phi = (\tau, z) \in \mathcal{E}^{\infty}_{H,Init}, \exists \delta > 0\ \left[t \in B(\delta, t_\infty) \Rightarrow x(t) \in B(\epsilon, \overline{x})\right],$$

for some chosen ϵ. Rewriting in linear temporal logic, this becomes

$$\forall \phi = (\tau, z) \in \mathcal{E}^{\infty}_{H,Init}\ \Diamond \left[x(t) \in B(\epsilon, \overline{x})\right]. \tag{8}$$

This property says that all trajectories eventually reach some set of the space described by a ball around $\overline{x}$.

In Equation (8), we have simply restricted (7) by considering only one ϵ, and so (7) $\Rightarrow$ (8). An equivalent statement of this is the contrapositive:

$$\neg(8) \Rightarrow \neg\,(7). \tag{9}$$

Now $\neg(8) \Leftrightarrow \exists \phi = (\tau, z) \in \mathcal{E}^{\infty}_{H,Init}\ \Box\left[x(t) \notin B(\epsilon, \overline{x})\right]$, which says that for (8) to be false we only require one infinite execution which never enters $B(\epsilon, \overline{x})$. So (9) says that finding one execution which disproves the liveness property $\Diamond\left[x(t) \in B(\epsilon, \overline{x})\right]$ will disprove Equation (7) which expresses global attractivity of $\overline{x}$.

So $\neg(8)$ requires at least one infinite execution to not satisfy the liveness property $\Diamond\left[x(t) \in B(\epsilon, \overline{x})\right]$. However, as mentioned before, finding infinite executions of hybrid automata is very hard, so we wish to define a property on finite executions which will imply that $\Diamond\left[x(t) \in B(\epsilon, \overline{x})\right]$ is not true for some set of infinite executions. If we can define and use such a property this will mean that we can disprove a global attractivity property by finding one finite execution.

In the next section, we will define deadness as such a property for disproving liveness properties with finite executions, and we will use it to disprove global attractivity of equilibrium points in the examples in Section 8.

5. Defining deadness

Once we have specified a desired liveness property, we wish to verify whether it is actually true, and this is where the complexity arises. Liveness properties are complex to verify, since it has to be shown that for all possible initial conditions and all possible executions from these initial conditions some property holds at some point in the future. If we cannot prove liveness, a sensible plan is to attempt to disprove it, and this could in turn disprove much more general properties, like global attractivity (as discussed in Section 4). However, even disproving liveness properties involves finding counterexample

executions of infinite length, as finite length executions could always be extended to something that satisfies the liveness property (by definition). Finding infinite length executions is especially hard in hybrid dynamical systems, due to the need to accurately represent a path in *continuous* space and time.

For this reason, we propose using another property alongside the liveness property which will disprove liveness with a finite execution if it is proved to be true. We call this property *deadness*: it is a concept related to dead states in automata theory. Note that, unlike safety and liveness, this property is related only to the valid executions of the hybrid automaton, not the more abstract future unaware versions.

Definition 13. A formula D on a hybrid automaton H is a *deadness property for liveness property* L iff any finite execution which satisfies D but not L cannot be extended to an infinite execution which satisfies L, or more formally

$$\forall \phi_{0,p} \in \mathcal{E}_H^F \big(\phi_{0,p} \vDash (D \wedge \neg L) \;\Rightarrow\; \forall \phi \in \mathcal{E}_H^\infty \left(\phi_{0,p} \prec \phi \;\Rightarrow\; \phi \nvDash L\right)\big). \tag{10}$$

We can define a deadness property for H with respect to the initial set *Init* by taking instead $\phi_{0,p} \in \mathcal{E}_{H,Init}^F$ and $\phi \in \mathcal{E}_{H,Init}^\infty$ in (10). ■

Intuitively, we are formalising the idea that 'when we are dead, we cannot be alive again'. As deadness is a property that is defined on finite executions, we must evaluate $\phi_{0,p} \vDash (D \wedge \neg L)$ with the chattering extension defined in Definition 8, that is $\phi_{0,p} \vDash (D \wedge \neg L) \;\equiv\; \phi_{0,p}^c \vDash (D \wedge \neg L)$.

From their definition, deadness properties are those which can be satisfied by a finite execution, or more formally by the chattering infinite execution of this execution (Definition 8). The idea is that once some point in space has been reached, the deadness property becomes true and would not become false again if the execution always remained at this point. Some properties which could be deadness properties (given in metric temporal logic (MTL)) are $\Diamond P$ (eventually the set described by P is reached), $\Diamond_{[0,\bar{t}]} P$ (within $\bar{t}$ seconds a set is reached), and $\Diamond\Box_{[0,\bar{t}]} P$ (eventually the set P is reached and the trajectory then remains there for $\bar{t}$ seconds).

We now define the way that the definition of deadness properties translates into descriptors for executions in the hybrid automaton.

Definition 14. A finite execution $\phi_{0,p} \in \mathcal{E}_U^F$ is called *dead* if it satisfies the deadness property D but not the liveness property L, or $\phi_{0,p} \vDash (D \wedge \neg L)$. A hybrid automaton H is said to be *dead* if there exists at least one finite execution of H which is dead after starting in *Init*, that is

$$H \text{ is dead} \Leftrightarrow \exists \phi_{0,p} \in \mathcal{E}_{H,Init}^F \big(\phi_{0,p} \vDash (D \wedge \neg L)\big). \quad \blacksquare \tag{11}$$

Lemma 1. *If the hybrid automaton H is dead with respect to a liveness property L and a deadness property D, then H is not live.* ■

Proof. Assume there exists $\phi_{0,p} = (\tau_{0,p}, z) \in \mathcal{E}_{H,Init}^F$ such that $\phi_{0,p} \vDash (D \wedge \neg L)$. Then for all extended executions $\phi \in \mathcal{E}_{H,Init}^\infty$, $\phi \nvDash L$ or $\phi_{0,p} \not\prec \phi$ by Definition 13. As $\phi_{0,p}$ is a finite execution, it is not maximal in H, and so (by non-blocking of H) there must be at least one infinite execution ϕ' such that $\phi_{0,p} \prec \phi'$, so $\phi' \nvDash L$. Now, this is a contradiction to the required condition for liveness of H in Definition 12, so the hybrid automaton H is not live. □

It is important to understand that the concept of deadness only tells us about liveness if it is proven — if deadness does not hold then this does not prove whether H is live or not. We are introducing a framework to help prove more instances of properties on hybrid systems, as using a deadness property can increase our ability to find counterexamples to liveness properties, by only requiring finite executions of the hybrid automaton to be found. We can use our dynamical knowledge of the hybrid automaton to create these deadness properties, so that finite executions are enough to disprove liveness — this is a new idea for formal verification of hybrid automata.

6. Finding deadness properties for hybrid systems

In this section, we will give one method for how deadness properties can be found, using dynamical properties of the hybrid system we are interested in. The idea is to find invariant sets which trap the executions of the hybrid automaton away from where the desired behaviour occurs, and then the deadness property is to show that such an invariant set is reached in finite time. Towards this goal, we give the definition of a hybrid invariant set.

Definition 15. A hybrid set $W \subseteq \mathcal{Z}$ is a *hybrid invariant set* for hybrid automaton H iff all executions starting in W remain there for all time, or more formally

$$\forall \phi = (\tau, z) \in \mathcal{E}_{H,W}\; \forall t \in \tau\; (z(t) \in W). \quad \blacksquare$$

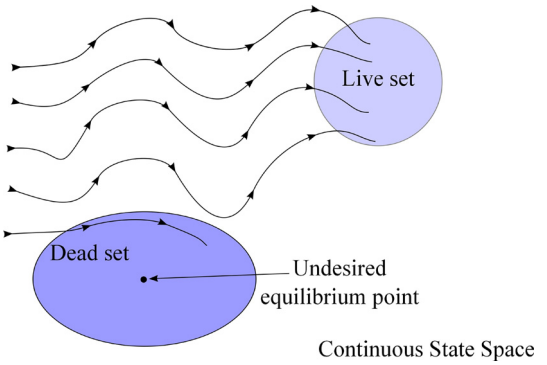

Fig. 3. Proving that the live set $\overline{L}$ is not reached by every execution, using a deadness property based around an equilibrium point. For clarity, only the continuous space is shown here.

A hybrid invariant set is a part of the hybrid state space $\mathcal{Z}$ which the executions of the system cannot leave once they have entered it. As a hybrid invariant set can include parts of more than one discrete location, executions which are trapped inside the set can still make both discrete and continuous transitions.

The method we propose finds a deadness property for a liveness property that says we reach some desired hybrid set $\overline{L} \subseteq \mathcal{Z}$ in the space, which is an inevitability property [18]. We will refer to this desired set $\overline{L}$ as the *live set*. The deadness property is to reach any of a selection of invariant sets which trap the dynamics away from $\overline{L}$, which is also an inevitability property. Fig. 3 shows the idea of this property in a visual way.

We make use of the fact that hybrid dynamical systems can have multiple equilibrium points, some of which we wish to tend to and some we want to avoid [54]. The algorithm we present does not consider more general types of limiting behaviour that can create invariant sets, although these could be added with different methods for finding the invariant sets. The other kinds of limiting behaviours are discussed in the conclusion of this work. We use here the notion of equilibria in *hybrid-space*, so that a point $(\overline{q}, \overline{x})$ can be an equilibrium of the hybrid automaton rather than a state vector $\overline{x}$ simply in continuous space.

Definition 16. $\overline{z} = (\overline{q}, \overline{x}) \in \mathcal{Z}$ is a *hybrid-space equilibrium* of the hybrid automaton H if both of the following conditions hold:

1. $\overline{x} \in Dom(\overline{q})$ and $f_{\overline{q}}(\overline{x}) = 0$.
2. $\overline{x} \notin G(e)$ for any $q \in Q$ such that $e = (\overline{q}, q) \in E$. ■

These hybrid-space equilibria effectively consider the dynamics in each location separately, only allowing a hybrid state to be considered an equilibrium if no continuous or discrete dynamics can change the value of an execution that starts at that hybrid state. This notion of equilibrium should be contrasted with the typical notion used in switched and hybrid systems, where equilibria are defined as points in continuous-space only, common to all locations where they exist. More formally, a continuous-space equilibrium $\overline{x} \in \mathbb{R}^n$ is defined as having (1) $f_q(\overline{x}) = 0$ for every $q \in Q$ where $\overline{x} \in Dom(q)$, and (2) if $\overline{x}$ ever occurs in a guard condition then it must be reset to itself (although the discrete location could change).

The continuous-space definition of equilibrium has evolved in the theory of stability of switched systems, where switching could happen at any time in the system. In such a context the only sensible idea of equilibrium will allow discrete motion to happen at an equilibrium point, as switching could happen whilst at the point. However, as we are considering the class of hybrid systems for which hybrid automata are a natural representation, and not the class of switched systems, it makes just as much sense to consider equilibria that can trap executions of the hybrid automata in one location only. When we consider these hybrid-space equilibria we can use the methods of Lyapunov functions for continuous systems to analyse the dynamics of each location. In particular, we can find invariant sets of each set of discrete dynamics, which will be invariant sets of the hybrid automaton if they do not intersect with any guard conditions.

The method we propose for finding deadness properties on a hybrid system described by a hybrid automaton is given in Algorithm 1. In particular, for each location $q \in Q$ of the hybrid automaton, the algorithm first finds all the equilibrium points it can, disregarding any equilibrium point in the desired set $\overline{L}$. Then the unstable equilibria are disregarded: for locations with nonlinear dynamics only locally asymptotically stable equilibria are kept, and for linear dynamics all stable equilibria are kept.[2] Every guard condition that allows executions to leave q is then tested to see if $\overline{x}$ can satisfy it, and if so then $\overline{x}$ is disregarded. We are then left only with equilibria of the type of Definition 16, with all these equilibria locally stable or asymptotically stable.

[2] These conditions are because we follow Lyapunov's indirect method, where we can only ensure that the stability of the nonlinear system is the same as that of the linearised system if the equilibrium is asymptotically stable. That is, the real parts of the eigenvalues are strictly less than zero.

Algorithm 1 Finding dead sets to disprove that all executions reach $\overline{L}$.

Input: Hybrid automaton H, and a live set $\overline{L} \subseteq \mathcal{Z}$ we would like to reach.
Output: A hybrid set $W \subseteq \mathcal{Z}$ which, if reached, will disprove the liveness property $\Diamond\overline{L}$.

```
1: W ← ∅ (initialise the hybrid dead set)
2: for all q ∈ Q do
3:     EQ_q ← find all solutions of equation f_q(x) = 0 in set Dom(q)
4:     for all x̄ ∈ EQ_q do
5:         if x̄ ∈ L̄ then
6:             remove x̄ from EQ_q
7:             continue (to next x̄)
8:         else if f_q is nonlinear and x̄ is not locally asymptotically stable then
9:             remove x̄ from EQ_q
10:            continue (to next x̄)
11:        else if f_q is linear and x̄ is unstable then
12:            remove x̄ from EQ_q
13:            continue (to next x̄)
14:        else
15:            for all e ∈ E, with e = (q, p) for any p do
16:                if x̄ ∈ G(e) then
17:                    remove x̄ from EQ_q
18:                    break loop (and go to next x̄ in EQ_q)
19:                end if
20:            end for
21:        end if
22:        if f_q is nonlinear then
23:            f̄_q ← linearised dynamics of f_q around x̄ in domain Dom(q)
24:        else
25:            f̄_q ← f_q
26:        end if
27:        V ← Lyapunov function for f̄_q
28:        W_V ← invariant set of dynamics f̄_q created from V
29:        W ← add (q, W_V) to the set of dead sets
30:    end for
31: end for
```

For each hybrid-space equilibrium point, the method then proceeds to find a Lyapunov function[3] for the linearised dynamics about this equilibrium point, which is then optimised to be a Lyapunov function for the nonlinear dynamics. This creates an invariant set in the continuous space $W_V \subseteq \mathcal{X}$ which traps all trajectories close to the equilibrium $\overline{x} \in \mathcal{X}$ which exists in location $q \in Q$. We can add this continuous invariant set to a hybrid invariant set by $W = W \cup (q \times W_V)$, and this will create a larger hybrid invariant set W.

Repeating this process in every location $q \in Q$ around each of the stable equilibrium points gives us a hybrid invariant set which, if reached, disproves the liveness property. Therefore, the deadness property is reaching the hybrid invariant set W.

7. Implementing the method to disprove liveness

We will now discuss how we have implemented Algorithm 1 to automatically find dead sets and how deadness can then be proved. We show how each part has been implemented and also discuss other methods for achieving the same ends. Most of the implementation has been made using MATLAB and its Symbolic Math Toolbox.[4]

The input to the implementation is given as a MATLAB structure describing the hybrid automaton in terms of the properties of each location (domain, dynamics, initial set and live set) and the properties of each transition (guard and reset).

7.1. Finding the stable equilibria in each location of H (lines 3–21)

It is a relatively simple task to find the equilibria of a set of dynamics, as we must just solve the equation $f_q(x) = 0$ for values of x. There are numerical methods for finding such equilibria, which are typically iterative optimisation algorithms like steepest descent minimisation methods. As we would like to find all of the equilibrium points, it makes sense to solve the equations symbolically, as would be done by hand. For this purpose, we have used the Symbolic Math Toolbox from MATLAB, but other symbolic mathematics engines could be used (Mathematica or Maple, for instance).

[3] With an abuse in the use of Lyapunov stability terminology, we will consider that our 'Lyapunov function' $V(x) : \mathbb{R}^n \to \mathbb{R}^n$ for a stable equilibrium $\overline{x}$ will have the properties that $V(x) > 0$ for all $x \in \mathbb{R}^n \setminus \{\overline{x}\}$, $V(\overline{x}) = 0$, and $\dot{V}(x) = \frac{dV}{dx}\dot{x} \leq 0$ for all $x \in \mathbb{R}^n \setminus \{\overline{x}\}$, where $\dot{x}$ denotes the time derivative of x.

[4] The programs used in this paper are part of the DeadRegions Toolbox available at http://staff.cs.manchester.ac.uk/~navarroe/research/dyverse/liveness/.

For each location q, we solve the equation $f_q(x) = 0$ to get possible equilibria $\overline{x} \in EQ_q$ using the symbolic *solve* function from the Symbolic Math Toolbox. We use the symbolic *subs* function to substitute the found equilibria into (1) the domain equation, to check the equilibrium is in the domain, (2) the guard conditions, to check it is not in a guard, and (3) the live set $\overline{L}$, to check it is not a desired point. If the system is nonlinear, for each equilibrium $\overline{x}$ we linearise f_q around $\overline{x}$ in domain $Dom(q)$ (by obtaining the associated Jacobian matrix of f_q). We then test the eigenvalues of the Jacobian of f_q at $\overline{x}$ to find the stability of the point, and remove those not locally stable — the ones with some eigenvalue ≥ 0. This uses the symbolic MATLAB functions *Jacobian* and *eig*, along with a linearity test for the dynamics. We get left with sets of locally stable equilibria EQ_q for each $q \in Q$. We highlight that the fact of using the linearisation of the nonlinear system to state the local stability of its equilibrium points poses restrictions on finding deadness properties, since some locally stable equilibrium points might be dismissed.

7.2. Creating an invariant set around a stable equilibrium point (lines 27–28)

The method we use for creating an invariant set around a given locally stable equilibrium point is due to [22], and relies on finding a quadratic Lyapunov function for this equilibrium point. There are various other methods available for finding Lyapunov functions and regions of attraction numerically in continuous systems (for instance [29,55,64,72]), but we selected the method in [22] because it can deal with arbitrary nonlinear systems to create a simple quadratic Lyapunov function. The method is made up of four parts: (1) a Lyapunov function is found for the system, which is linearised about the equilibrium point, if the system is nonlinear; (2) this function is given an arbitrarily small radius, and then the radius is optimised so it has as large an area as possible; (3) the quadratic Lyapunov function itself is optimised to have the largest area possible for the dynamics using the result of step (2) as the starting point; then (4) the resulting function is checked with a very fine mesh to make sure it is actually a Lyapunov function, and if not we start from step (2) again, using a finer mesh of vectors for the optimisation. We now give a closer look at the implementation for each of these steps.

1. Solving for a Lyapunov function of a linear vector field $\dot{x} = Ax + b$ involves solving the *Lyapunov equation*, $A^T P + PA = -Q$, where Q is chosen and positive definite, and P is positive definite. This gives us a Lyapunov function for this linearisation of the form $V(x) = (x - \overline{x})^T P (x - \overline{x})$. There are many algorithms available to solve such Lyapunov equations, for example [37], and the *Lyap* method from the Control System Toolbox in MATLAB. We use the highly efficient method `f08qh` from the NAG toolbox for MATLAB.
2. Given this $V(x)$ from the first step, we start with $V(x) < \epsilon$ for some small ϵ — we have used $\epsilon = 10^{-5}$. We then used `fminsearchcon`, a constrained optimisation method obtained from the MATLAB file exchange, to expand this to a larger region $V(x) < \epsilon_1$. This optimisation is made subject to the larger set still being a Lyapunov function for the nonlinear dynamics, and not intersecting the guards or the live set $\overline{L}$. This optimisation is achieved through using randomised vectors spanning the space, and so the set found will be slightly different every time the algorithm is run.[5]
3. We can then try to find a more optimal trapping set created by a new Lyapunov function $V_1(x) = (x - \overline{x})^T P_1 (x - \overline{x}) < 1$, using $V(x) < \epsilon_1$ as a starting point for the optimisation. The optimisation is made over the area of the set enclosed by the equation $V_1(x) < 1$, with the same constraints as for step 2.
4. The boundary $V_1(x) = 1$ is then tested to make sure that the flow across the boundary is always inwards, and it is also checked that it does not intersect with the guard conditions or live set. This is achieved by testing the values of the functions at every point of a very fine mesh over the surface.

We should briefly discuss where numerical errors could creep in to this method. The first place is at the beginning of step 2 of the optimisation, where the Lyapunov function is given a very small radius. The correctness of this step depends on $V(x) = \epsilon$ being a trapping region for the nonlinear system: if $\dot{V}(x) > 0$ anywhere on the surface $V(x) = \epsilon$ then this is not a trapping set for the nonlinear dynamics. We have selected a very small epsilon which should be fine for most systems, but automatic checks can be built to make sure that this initial set is actually an invariant set for the nonlinear dynamics, and ϵ can be reduced accordingly if not.

The second place that numerical errors can affect the method is in the optimisation, where we rely on having a mesh of vectors over the surface of the Lyapunov function to make sure we are optimising within the region where the function creates an invariant set for the nonlinear dynamics. If there are not enough vectors then important bumps and spikes in the flow could be missed and the optimisation could continue even though the condition $\dot{V}(x) \leq 0$ has been breached. However, this problem is reduced by step 4 of the method, where the optimisation is repeated with more vectors if it does not create a suitable Lyapunov function after one run-through. [22] suggests suitable numbers of vectors to start the optimisation with for different dimensional systems, which we have used in our implementation.

[5] The set can be checked to be a valid invariant by checking the value of the derivative on the boundary using a theorem prover such as MetiTarski [2], although this has not been implemented.

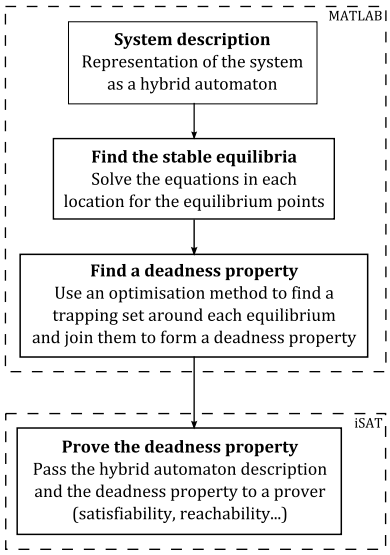

Fig. 4. An overview of the method to disprove liveness by proving deadness in a hybrid system.

7.3. Proving the deadness property

In order to prove the deadness property, we need to find at least one execution of the hybrid automaton which eventually reaches one of the dead sets. This is effectively a *satisfiability* (SAT) problem on the hybrid system: SAT problems are of the form 'given a system and a logical specification, is there an execution of the system which satisfies the specification?' We have the SAT problem 'given our hybrid automaton H and the specification 'eventually reach a dead set', is there an execution of H which satisfies the specification?'

In the domain of computer science, specialised solvers are used for finding a solution of a SAT problem, and such solvers are many and varied in type. In the domain of hybrid systems and real-arithmetic SAT solving, there are only a few solvers available, including iSAT [30] and ABsolver [10]. Of these, iSAT is the most well developed for hybrid systems and has an easy-to-use input method, so we make use of this solver in our work (see Fig. 4). It is based on bounded model checking (BMC), relying on a fixed time-step discretisation of the dynamics of the hybrid automaton, so that on the k-th iteration iSAT will look at all executions of length $k\Delta t$, where Δt is the length of the time step.

In our framework, iSAT performs BMC by first allowing trajectories of $0 \cdot \Delta t$ length, then $1 \cdot \Delta t$ length, etc., to attempt to get a valid execution stopping at the target dead zone. It stops when either it finds such a trajectory or it finds one where the intervals it works in are too small to be split again. In this case, it can decide that the trajectory is 'unknown' in terms of its satisfiability of the required deadness condition, but this trajectory could actually have reached the dead zone. iSAT will keep trying run lengths in terms of numbers of steps up to the maximum specified in the inputs.

The input to iSAT is a file containing a representation of the hybrid automaton, with a fixed-step discretisation of the continuous dynamics. For example, for the case study presented in the next section, we use the forward Euler's method.

The file also defines a target region for the automaton executions — for our case it will be a mixed logic and inequality constraint specifying the dead sets that have been calculated by the implementation of Algorithm 1. The possible outputs for iSAT are 'unsatisfiable', 'satisfiable' (with the satisfying solution returned), and 'unknown' (with a candidate solution returned). The candidate solution offered by iSAT when it returns with status 'unknown' could still be a satisfying execution, but the 'highly incomplete deduction calculus' [41] means that iSAT cannot always decide whether it is.

It is worth noting that although we choose the discretisation time step, iSAT overapproximates the trajectory for this time step so that the choice of the time step should not affect the proving capabilities of the system. iSAT guarantees the overapproximation is appropriate to ensure that if a candidate solution is found then we are certain it has followed from the initial starting condition. Therefore if iSAT produces a trajectory that satisfies the condition for the dead region then we assume we have found a suitable dead trajectory and we prove the deadness condition.

Using a fixed time-step discretisation of the dynamics can create problems with fast-changing dynamical systems (whether continuous or hybrid), as a large numerical error can be built up as iSAT approximates the flow. This is a problem which can only be helped by using fewer or smaller time-steps — we will look at some examples which work in the example of the drillstring in the next section.

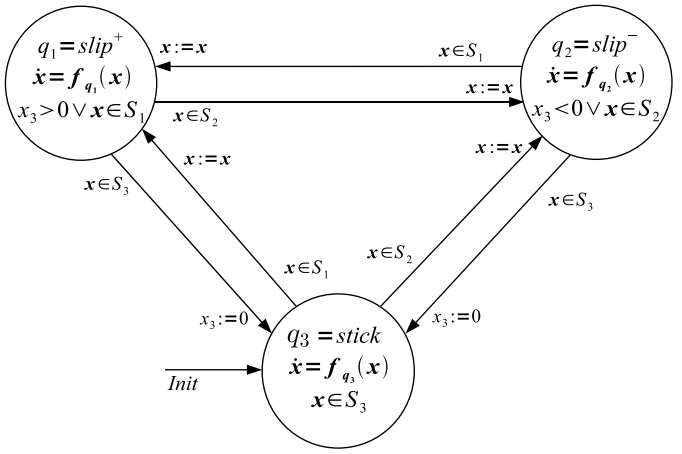

Fig. 5. The hybrid automaton for the drillstring example [51].

8. Application of the methodology in the examples

8.1. The drillstring example

As we saw in Section 2, the drillstring model has three possible long-term behaviour patterns, driven by two equilibrium points that can be locally attractive. The 'strength' of the attractivity of each equilibrium point is dependent on two parameters: the driving torque at the top of the drill u, and the weight on the bit W_{ob}. We wish to tend to the locally attractive positive velocity equilibrium so that the drill is constantly making progress into the ground.

We consider the hybrid automaton model of the drillstring given in Fig. 5. All the continuous dynamics have the form of equation (1), with only the term T_{f_b} changing with the location as defined below (see equations (2) and (3) for definitions of f_b and u_{eq}):

$$T_{f_b}(x) = \begin{cases} f_b(x_3) & \text{if } q = q_1, \\ -f_b(x_3) & \text{if } q = q_2, \\ u_{eq}(x) & \text{if } q = q_3. \end{cases}$$

For ease of notation, we also use the following three sets:

$$S_1 = \{x \in \mathbb{R}^3 : (x_3 > 0) \vee (x_3 = 0 \wedge u_{eq}(x) > T_{sb})\},$$
$$S_2 = \{x \in \mathbb{R}^3 : (x_3 < 0) \vee (x_3 = 0 \wedge u_{eq}(x) < -T_{sb})\},$$
$$S_3 = \{x \in \mathbb{R}^3 : (x_3 = 0) \wedge (|u_{eq}(x)| \leq T_{sb})\}.$$

We will look at disproving the property of global attractivity of the positive velocity equilibrium. Global attractivity being true implies that the liveness property 'a small sphere $\overline{L}$ around the positive velocity equilibrium is eventually reached' is also true. In fact, this liveness property is an inevitability property. Hence, we can use Algorithm 1 to find deadness properties in order to prove that, for some selection of parameters, we will never get close to the desired equilibrium. This will imply that the equilibrium is not globally attractive. We have tested our prototype implementation of the algorithm on this example with values of $W_{ob} = 59000$ N and $u = 6000$ N m: from simulation we expect that the executions will have the 'stuck' behaviour for this values of W_{ob} and u (i.e. be attracted to the undesired equilibrium).

The algorithm analyses each location in turn, stating with q_1, the positive velocity location. We give a summary of the results of the implementation.

- Analyse $f_{q_1}(x) = 0$ in the set $Dom(q_1) = S_1$. Find two equilibrium points at $\overline{x}_1 = \overline{x}_3 > 0$ as the two solutions of the equations $u - (c_r + c_b)\overline{x}_3 - f_b(\overline{x}_3) = 0$, and $\overline{x}_2 = (u - c_r\overline{x}_3)/k_t$. The first one found is at $(2.07, 5.94, 2.07)^T$ and is unstable in the linearisation, and the second one is at $\overline{x}_{q_1} = (1.62, 6.17, 1.62)^T$. Define the live set as $\overline{L} = \{x \in Dom(q_1) : J_r(x_1 - \overline{x}_{q_1,1})^2 + k_t(x_2 - \overline{x}_{q_1,2})^2 + J_b(x_3 - \overline{x}_{q_1,3})^2 < 1\}$. The equilibrium $\overline{x}_{q_1}$ is the desired equilibrium inside the specified live set $\overline{L}$, so move on.
- Analyse $f_{q_2}(x) = 0$ in the set $Dom(q_2) = \{x \in \mathbb{R}^3 : (x_3 < 0) \vee (x_3 = 0 \wedge u_{eq}(x) < -T_{sb})\}$. Find no solutions of the equation with the current parameters, and so move on.
- Analyse $f_{q_3}(x) = 0$ in the set $Dom(q_3) = \{x \in \mathbb{R}^3 : (x_3 = 0) \wedge (|u_{eq}(x)| < T_{sb})\}$. Find an equilibrium point at $\overline{x}_1 = \overline{x}_3 = 0$ and $\overline{x}_2 = u/k_t = 6.96$. It is not the desired equilibrium, but it is stable for the chosen parameter values. It is not in any of the guards from location q_3 for our parameters, so we want to find an invariant set around it to use as a dead set.

- In location q_3, we have 'lost a dimension', due to $x_3 = 0$ always being true, with dynamics $\dot{x}_3 = 0$. Our implementation automatically detects this by looking at the conditions for the domain (in particular $x_3 = 0$), and as the dynamics are linear, we can allow the resulting zero eigenvalue whilst keeping stability in the x_1–x_2 plane. Let us write $y = (x_1, x_2)^T$ for the new variables after projection into the $x_3 = 0$ plane. The implementation then solves the Lyapunov equation for the restricted dynamics of location q_3 to get a first guess at a Lyapunov function:

$$V(y) = (y - \overline{y})^T \begin{pmatrix} 6.15 & 1.23 \\ 1.23 & 2.84 \end{pmatrix} (y - \overline{y}),$$

  where $\overline{y} = (0, 6.96)^T$. Although this function would not prove attractivity of the equilibrium point in 3-dimensions, we can use it to provide a trapping set for the executions, which is what we are actually interested in.
- Starting from $V(y) < 10^{-5}$, extend this to a larger set by using the optimisation method of [22] which we have implemented, taking the domain of q_3 and the non-satisfaction of the guards on edges out of q_3 as extra constraints. Our method gives an invariant set of

$$W_{q_3} \equiv \left\{ y : (y - \overline{y})^T \begin{pmatrix} 0.652 & 0.110 \\ 0.110 & 0.410 \end{pmatrix} (y - \overline{y}) < 1 \right\},$$

  in location q_3, which forms a hybrid invariant set of $W = \{(q, x) \in \mathcal{Z} : q = q_3,\ (x_1, x_2)^T \in W_{q_3},\ x_3 = 0\}$ to be the dead set for the hybrid automaton.[6] In terms of the termination of the computation of a larger invariant set, the termination properties are the same as in the underlying optimisation algorithm of [22].

Our algorithm has created a deadness condition (reaching an invariant set in q_3), which, if we can find at least one execution from the initial set *Init* to satisfy it, will disprove the liveness property of all executions of the hybrid automaton tending to the desired equilibrium.

We have attempted to find a satisfying execution using iSAT [30]. For the drillstring example, iSAT returns 'unknown' for all initial conditions tried, however close or far away from the dead set they are. This may be to do with the fact that the dead set we want to reach is in a lower-dimensional subspace, and it is therefore difficult to numerically check that the candidate solution offered is allowable. However, in some cases where iSAT returns 'unknown', the candidate solution offered actually is a satisfying dead execution, which we can check by substituting the last time point of the candidate solution into the equation for the dead set which we are attempting to satisfy. These cases that return valid candidate solutions are typically achieved by choosing an initial condition close to the dead set (in terms of time separation), which allows us to specify a much smaller time step.

For instance, choosing an initial condition of $x_1 = 0.09$, $x_2 = 8.5$, $x_3 = 0.0002$ and $q = q_1$, with time step $\Delta t = 0.01$ s iSAT finds a dead execution in 11 steps (1 discrete transition, then 10 time steps), ending at a set of points of $x_1 \in (0.02580196, 0.02580197)$, $x_2 \in (8.50609831, 8.50609832)$, $x_3 \in [0, 0]$ and $q = q_3$. Substituting these values into W, the equation for the invariant set, we find that these last points satisfy W, and so we can conclude that this is a dead execution, and so the hybrid automaton H is dead for this initial condition. This is an interesting result, because (with these parameters) we know that the drillstring cannot recover from such a dead execution, and so cannot obtain the desired behaviour of getting close to the desired equilibrium. Consequently, we know with certainty that there are some initial states which we should not use to start execution of the system, and can avoid them in order to satisfy the liveness property that we reach the desired equilibrium.

For a more general result, we specify an initial set by taking intervals around the point initial condition given above, so that $x_1 \in (0.08, 0.1)$, $x_2 \in (8, 9)$, $x_3 \in (0.0001, 0.0003)$ and $q = q_1$. Then, with the same time step, iSAT finds the shortest execution (in number of steps) which starts in this initial set and reaches the set W: actually iSAT finds an execution which makes only one discrete transition and no time steps, starting anywhere in set with $q = q_1$, $x_1 \in [0.09749999, 0.09859424]$, $x_2 \in [8.02374976, 9.0240449]$, $x_3 \in [0.00009999, 0.00030001]$, and ending anywhere in the set with $q = q_3$, $x_1 \in [0.09749999, 0.09875001]$, $x_2 \in [8.02374976, 9.0240449]$, $x_3 \in [0, 0]$, which is inside the dead set W. Again this is returned as 'unknown' but checking the result by substitution into W shows that it is a dead execution, and we can conclude that the hybrid automaton H is dead for this set of initial states. This is even more helpful than the previous result, as we have obtained a particularly bad execution which can satisfy the deadness condition with one discrete transition, which is an initial point we should definitely avoid.

8.2. The discontinuous Lorenz system

The dynamical behaviour of the Lorenz system can be reproduced by a discontinuous system, which is referred to as *piecewise-linear* (PWL) *Lorenz system* and has the form [7,8,47]:

[6] When these results are replicated, slightly different results will be obtained due to the randomised allocation of vectors that are used in the optimisation.

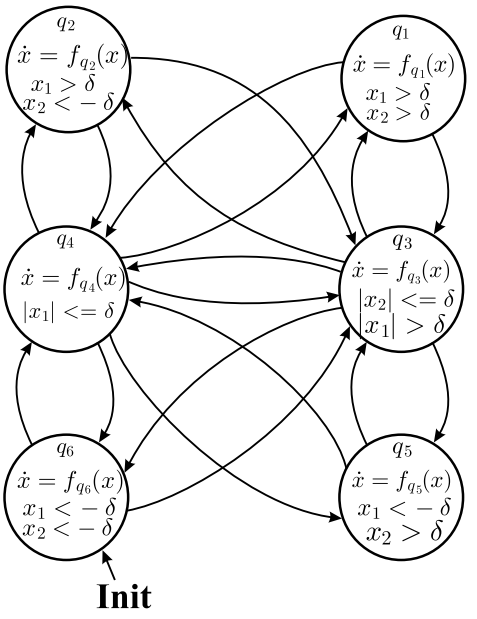

Fig. 6. Graphical representation of the hybrid automaton for the piecewise-linear Lorenz system proposed in [50].

$$\begin{aligned}\dot{x}_1 &= a(x_2 - x_1),\\ \dot{x}_2 &= \text{sign}(x_1)(\rho - x_3) - qx_2,\\ \dot{x}_3 &= \text{sign}(x_2)x_1 - bx_3.\end{aligned} \tag{12}$$

As the work [50] proposes, system (12) can be modelled by a hybrid automaton that is referred to as the Lorenz hybrid automaton H_L and is given in Fig. 6. The main elements of H_L are the following ones [50]:

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$. $\mathcal{X} \subseteq \mathbb{R}^3$, $x = (x_1, x_2, x_3)^T$.
- $Dom(q_1) = \{x \in \mathbb{R}^3 : x_1 > \delta, x_2 > \delta\}$, $Dom(q_2) = \{x \in \mathbb{R}^3 : x_1 > \delta, x_2 < -\delta\}$, $Dom(q_3) = \{x \in \mathbb{R}^3 : |x_1| > \delta, |x_2| \leq \delta\}$, $Dom(q_4) = \{x \in \mathbb{R}^3 : |x_1| \leq \delta\}$, $Dom(q_5) = \{x \in \mathbb{R}^3 : x_1 < -\delta, x_2 > \delta\}$, $Dom(q_6) = \{x \in \mathbb{R}^3 : x_1 < -\delta, x_2 < -\delta\}$.
- Dynamics for each discrete location:

$$f_{q_1} = \begin{pmatrix} a(x_2 - x_1) \\ \rho - x_3 - qx_2 \\ x_1 - bx_3 \end{pmatrix}, \; f_{q_2} = \begin{pmatrix} a(x_2 - x_1) \\ \rho - x_3 - qx_2 \\ -x_1 - bx_3 \end{pmatrix}, \; f_{q_3} = \begin{pmatrix} -ax_1 \\ 0 \\ 0 \end{pmatrix},$$
$$f_{q_4} = \begin{pmatrix} 0 \\ 0 \\ -bx_3 \end{pmatrix}, \; f_{q_5} = \begin{pmatrix} a(x_2 - x_1) \\ x_3 - qx_2 - \rho \\ x_1 - bx_3 \end{pmatrix}, \; f_{q_6} = \begin{pmatrix} a(x_2 - x_1) \\ x_3 - qx_2 - \rho \\ -x_1 - bx_3 \end{pmatrix}.$$

- $G(q_3, q_1) = G(q_4, q_1) = Dom(q_1)$, $G(q_1, q_4) = G(q_5, q_4) = \{x \in \mathbb{R}^3 : |x_1| \leq \delta, x_2 > \delta\}$, $G(q_3, q_5) = G(q_4, q_5) = Dom(q_5)$, $G(q_1, q_3) = G(q_2, q_3) = \{x \in \mathbb{R}^3 : |x_2| \leq \delta, x_1 > \delta\}$, $G(q_3, q_4) = \{x \in \mathbb{R}^3 : |x_1| \leq \delta, |x_2| \leq \delta\}$, $G(q_5, q_3) = G(q_6, q_3) = \{x \in \mathbb{R}^3 : |x_2| \leq \delta, x_1 < -\delta\}$, $G(q_3, q_2) = G(q_4, q_2) = Dom(q_2)$, $G(q_2, q_4) = G(q_6, q_4) = \{x \in \mathbb{R}^3 : |x_1| \leq \delta, x_2 < -\delta\}$, $G(q_3, q_6) = G(q_4, q_6) = Dom(q_6)$, $G(q_4, q_3) = Dom(q_3)$.
- $R(q_i, q_j, x) = \{x\}$, $\forall i, j \in \{1, \ldots, 6\}$, and $i \neq j$.

The values for the parameters that we will use for our study are $b = 0.15$, $\rho = 7.0$, $q = 0.1$, $\delta = 10^{-6}$. We will explore two cases:

- $a = 1.2$: the system presents chaotic behaviour (Fig. 7).
- $a = 5.2$: the system trajectories converge to one of the equilibrium points of the system.

For both cases ($a = 1.2$, $a = 5.2$), we identify four equilibrium points:

- In location q_1: $\overline{x}_{q_1} = (1.034483, 1.034483, 6.896552)^T$.
- In location q_2: $\overline{x}_{q_2} = (-1.06599, -1.06599, 7.1066)^T$.

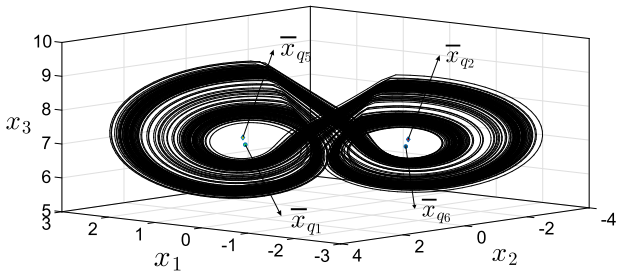

Fig. 7. Chaotic attractor of the Lorenz hybrid automaton for $a = 1.2$ with the equilibrium points for locations q_1, q_2, q_5 and q_6. The chaotic attractor is obtained with a simulation time of 1800 s and initial conditions $q_0 \times x_0 = q_6 \times (-2.1 \cdot 10^{-4},\ -5.20625 \cdot 10^{-3},\ 6.895783)^T$. Only the continuous state space $\mathcal{X}$ of the hybrid automaton is shown.

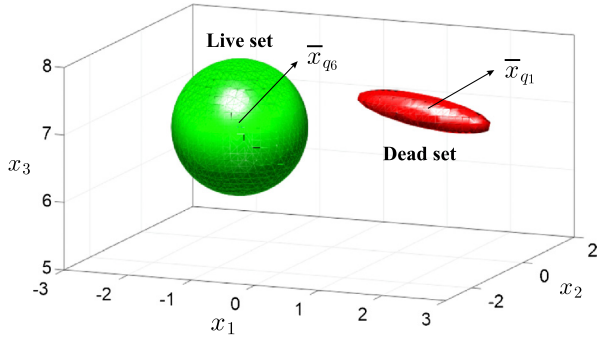

Fig. 8. Results of Algorithm 1 in the continuous state space to disprove the global attractivity of $\overline{x}_{q_6}$ for the Lorenz hybrid automaton when $a = 5.2$. Live set defined (green) in location q_6 and dead set found (red) in location q_1. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

- In location q_5: $\overline{x}_{q_5} = (1.06599,\ 1.06599,\ 7.1066)^T$.
- In location q_6: $\overline{x}_{q_1} = (-1.034483,\ -1.034483,\ 6.896552)^T$.

Equilibria $\overline{x}_{q_2}$ and $\overline{x}_{q_5}$ are virtual because they do not belong to the domain of their corresponding discrete locations. A virtual equilibrium of a discrete location in a hybrid automaton H is defined in [54] as: $\overline{x}_{q_i} \in \mathbb{R}^n$ is a *virtual equilibrium* of location $q_i \in Q$ if $f_{q_i}(\overline{x}_{q_i}) = 0$ and $\overline{x}_{q_i} \notin \mathrm{cl}(Dom(q_i))$, but $\overline{x}_{q_i} \in \mathrm{cl}(Dom(q_j))$ for some $q_j \in Q$, $q_j \neq q_i$.

Equilibria $\overline{x}_{q_2}$ and $\overline{x}_{q_5}$ are unstable for $a = 1.2$ and $a = 5.2$.

For the case when the system presents chaotic behaviour (for example, $a = 1.2$), if we try to disprove the property of global attractivity of equilibria $\overline{x}_{q_1}$ or $\overline{x}_{q_6}$ and use Algorithm 1, no dead set can be created since both equilibria are unstable. More interesting results are obtained when $a = 5.2$. In this case, equilibria $\overline{x}_{q_1}$ and $\overline{x}_{q_6}$ are asymptotically stable. We choose equilibrium $\overline{x}_{q_6}$ and will apply our methodology to disprove the global attractivity of $\overline{x}_{q_6}$ for $a = 5.2$. The results are given below.

To disprove the global attractivity of $\overline{x}_{q_6}$, we define the live set $\overline{L} = \left\{x \in Dom(q_6) : (x - \overline{x}_{q_6})^T (x - \overline{x}_{q_6}) < 1\right\}$. Algorithm 1 finds an invariant set within the domain of location q_1 as:

$$W_{q_1} \equiv \left\{ x \in Dom(q_1) : (x - \overline{x}_{q_1})^T \begin{pmatrix} 1.1906 & -0.1076 & 2.2993 \\ -0.1076 & 15.9118 & -0.9753 \\ 2.2993 & -0.9753 & 17.1868 \end{pmatrix} (x - \overline{x}_{q_1}) < 1 \right\},$$

which forms a hybrid invariant set of $W = \{(q, x) \in \mathcal{Z} : q = q_1,\ (x_1, x_2, x_3)^T \in W_{q_1}\}$ to be the dead set for the hybrid automaton. The live and dead sets are given in Fig. 8. In the figure, we only present the continuous state space $\mathcal{X}$ of the Lorenz hybrid automaton H_L.

To conclude with, Algorithm 1 has given a useful output to better understand the dynamics of the Lorenz hybrid automaton and the properties of global attractivity of its equilibrium points.

8.3. The 2-tank system

We consider a model of two liquid holding tanks which are connected in series by a pipe, with the first tank located higher than the second tank. This system was first studied in the context of hybrid systems in [68] and can be represented by a hybrid automaton, which is shown in Fig. 9. This is an example of a nonlinear hybrid system that has become a standard benchmark problem for verification of hybrid systems (see [24] and references therein). To show the applicability

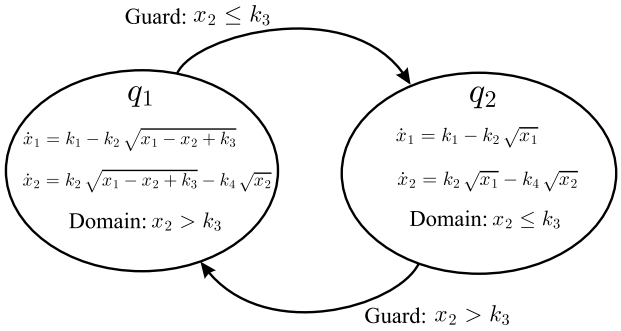

Fig. 9. Graphical representation of the hybrid automaton of the two-tank system with its main elements. $x = (x_1, x_2)^T$ where x_1 is the height of the water in the first tank, and x_2 is the height of the water in the second tank.

of our methodology, this example is especially interesting because it has virtual equilibria in both locations of the hybrid automaton.

By having a look at the domains of both locations in Fig. 9, we infer that depending on the values of the parameter k_3, the equilibrium points for each location may become virtual. We have chosen two case studies: $k_3 = 0.5$ and $k_3 = 1$. We consider for both cases $k_1 = 0.75$ and $k_2 = k_4 = 1$. We will conclude that the virtual equilibria of the discrete locations of the hybrid automaton cannot be globally attractive.

8.3.1. First case study: $k_3 = 0.5$

We have two locally asymptotically stable equilibrium points:

- For location q_1: $\overline{x}_{q_1} = (0.625,\ 0.5625)^T$.
- For location q_2: $\overline{x}_{q_2} = (0.5625,\ 0.5625)^T$. This equilibrium is virtual because it does not belong to $Dom(q_2)$ $(x_2 \leq k_3)$ but it belongs to $Dom(q_1)$ $(x_2 > k_3)$.

Consequently, for $k_3 = 0.5$, we cannot apply our algorithm to create a dead set for location q_2 when the live set is defined in location q_1. However, we can still apply our algorithm defining the live set around $\overline{x}_{q_2}$, even though it is a virtual equilibrium. In order to apply our algorithm, we need to find at least one locally asymptotically stable equilibrium outside the defined live set. For this example, we introduce an interesting concept of *virtual live set* around a virtual equilibrium. The results are the following ones:

- We define the live set around $\overline{x}_{q_2}$:

$$\overline{L} = \left\{ x \in \mathbb{R}^2 : (x - \overline{x}_{q_2})^T (x - \overline{x}_{q_2}) < R \right\},$$

  with $R = 0.001$. $\overline{L}$ is a virtual live set which might be outside $Dom(q_2)$ because it is defined around a virtual equilibrium which is outside $Dom(q_2)$.
- We solve the Lyapunov equation defined in Step 1 of Section 7.2 to get an initial guess at a Lyapunov function V (line 27 of Algorithm 1) before the optimisation algorithm of [22] is applied to obtain the dead set:

$$V(x) = (x - \overline{x}_{q_1})^T \begin{pmatrix} 1.5 & 0.75 \\ 0.75 & 0.75 \end{pmatrix} (x - \overline{x}_{q_1}).$$

- An invariant set is computed within location q_1 as:

$$W_{q_1} \equiv \left\{ x \in Dom(q_1) : (x - \overline{x}_{q_1})^T \begin{pmatrix} 42.2856 & -14.4901 \\ -14.4901 & 261.0237 \end{pmatrix} (x - \overline{x}_{q_1}) < 1 \right\},$$

  which forms a hybrid invariant set of $W = \{(q, x) \in \mathcal{Z} : q = q_1,\ (x_1, x_2)^T \in W_{q_1}\}$ to be the dead set for the hybrid automaton.

The results of Algorithm 1 are given in Fig. 10, where only the continuous state space of the hybrid automaton is shown. We can see that the dead set contains the virtual live set in the continuous space. However, it is important to note that these two sets are defined in the hybrid space, and they exist in different discrete locations of the hybrid automaton. So they do not intersect in the hybrid space.

Hence, we can use this dead set to disprove the liveness property that the live set is always reached, by showing that the trajectories of the system reach this dead set instead. This deadness property could now be shown using iSAT in a similar way to the drillstring example of Section 8.1.

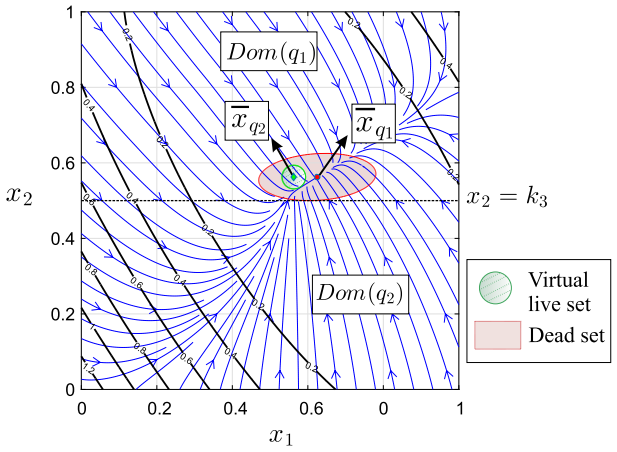

Fig. 10. Trajectories (blue lines with arrows) for different initial conditions in the continuous state space of the 2-tank system when $k_3 = 0.5$. Algorithm 1 is applied for a virtual live set (green striped area) defined around the virtual equilibrium $\overline{x}_{q_2}$, and a dead set is obtained within location q_1 (red shaded area) based on $\overline{x}_{q_1}$. The bold black ellipses represent the level sets of the initial Lyapunov function calculated before the optimisation suggested by [22] was applied to obtain the dead set. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

8.3.2. Second case study: $k_3 = 1$

We have two locally asymptotically stable equilibrium points:

- For location q_1: $\overline{x}_{q_1} = (0.125,\ 0.5625)^T$. This equilibrium is virtual because it does not belong to $Dom(q_1)$ $(x_2 > k_3)$ but it belongs to $Dom(q_2)$ $(x_2 \leq k_3)$.
- For location q_2: $\overline{x}_{q_2} = (0.5625,\ 0.5625)^T$.

With $k_3 = 1$, we cannot define the live set in location q_2 because, in this case, the dead set would be calculated based on a virtual equilibrium of q_1. In this situation:

- We define the live set around $\overline{x}_{q_1}$:

$$\overline{L} = \left\{ x \in \mathbb{R}^2 : (x - \overline{x}_{q_1})^T (x - \overline{x}_{q_1}) < R \right\},$$

  with $R = 0.01$. Here, $\overline{L}$ is also a virtual live set which might be outside $Dom(q_1)$ because it is defined around a virtual equilibrium which is outside $Dom(q_1)$.
- We solve the Lyapunov equation defined in Step 1 of Section 7.2 to get an initial guess at a Lyapunov function V (line 27 of Algorithm 1) before the optimisation algorithm of [22] is applied to obtain the dead set:

$$V(x) = (x - \overline{x}_{q_2})^T \begin{pmatrix} 1.125 & 0.375 \\ 0.375 & 0.75 \end{pmatrix} (x - \overline{x}_{q_2}).$$

- An invariant set is computed within location q_2:

$$W_{q_2} \equiv \left\{ x \in Dom(q_2) : (x - \overline{x}_{q_2})^T \begin{pmatrix} 0.00010985 & 0.025576418 \\ 0.025576418 & 6.55129366647 \end{pmatrix} (x - \overline{x}_{q_2}) < 1 \right\},$$

  which forms a hybrid invariant set of $W = \{(q, x) \in \mathcal{Z} : q = q_2,\ (x_1, x_2)^T \in W_{q_2}\}$ to be the dead set for the hybrid automaton.

The results of Algorithm 1 are given in Fig. 11, where only the continuous state space is shown. The results and conclusions are similar to the case of $k_3 = 0.5$. The dead set contains the virtual live set in the continuous space, but they do not intersect in the hybrid space.

These results have been obtained with the DeadRegions Toolbox that is available at http://staff.cs.manchester.ac.uk/~navarroe/research/dyverse/liveness/.

9. Conclusions and extensions of our work

In order to disprove liveness properties in hybrid automata a novel framework has been proposed, defining a new logical property called deadness. Deadness is a dynamically-aware property of the hybrid automaton which, if true, disproves the liveness property by means of a finite execution: we usually require an infinite execution to disprove a liveness property. An

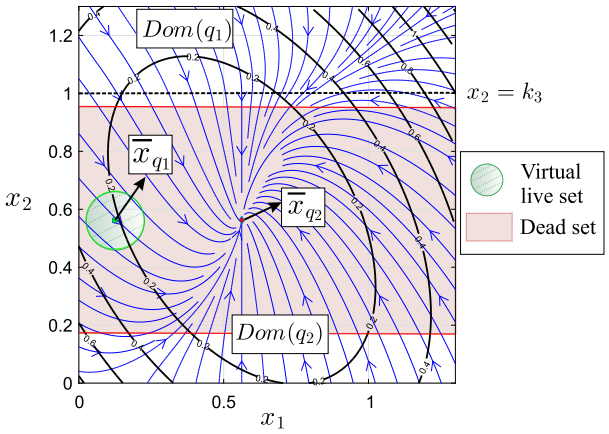

Fig. 11. Trajectories (blue lines with arrows) for different initial conditions in the continuous state space of the 2-tank system when $k_3 = 1$. Algorithm 1 is applied for a virtual live set (green striped area) defined around the virtual equilibrium $\overline{x}_{q_1}$, and a dead set is obtained within location q_2 (red shaded area) based on $\overline{x}_{q_2}$. The bold black ellipses represent the level sets of the initial Lyapunov function calculated before the optimisation suggested by [22] was applied to obtain the dead set. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

algorithm has been proposed which uses dynamical properties of hybrid systems to derive deadness properties automatically when proving inevitability properties, a class of liveness which says that eventually a set is reached. The algorithm uses invariant sets around undesired equilibria as 'dead sets' which disprove liveness if they can be proven to be reached. Since there are methods available to implement all of the steps of the algorithm, we have made a prototype implementation in MATLAB and iSAT (a hybrid system SAT solver) and have tested it on a simplified model of an oilwell drillstring. A dead set was calculated for this model, and the deadness condition was proven to hold for selected initial conditions using iSAT, although the resulting executions had to be checked manually. In addition, we have shown the applicability of the algorithm to find deadness properties in two more examples that represent two classes of hybrid systems with complex behaviours.

We have not solved the problem of finding dead sets for every type of behaviour that can occur in a hybrid automaton, so immediate extensions of our work may focus on other types of behaviour which can cause invariants in a hybrid system. Mainly:

1. Equilibrium points where the executions can keep jumping between locations whilst still at the same continuous point. For this kind of continuous-space equilibrium, using multiple or common Lyapunov functions [16,46] for the hybrid automaton may be more suitable for finding dead sets.
2. Trapping sets caused by stable periodic orbits. We have not discussed this kind of trapping set, as it is difficult to find periodic orbits in an automated way. However, there are methods which find such periodic orbits and their trapping sets [35], which could potentially be used to find more deadness conditions.
3. Convergence to an equilibrium point caused by discrete transitions (like that which occurs in switching controllers for electronic circuits [38]). This kind of stability is caused only by the existence of the discrete transitions, and is characterised by oscillating executions of the system which may (or may not) converge to an equilibrium.

The methods of proving such deadness properties can be also improved, in particular through better SAT solvers for hybrid systems, or through using already developed reachability algorithms.

Acknowledgements

The authors gratefully acknowledge the efforts of the anonymous reviewers, who gave valuable comments to improve the paper.

This work has been supported by the Engineering and Physical Sciences Research Council (EPSRC) of the UK under the framework of the project *DYVERSE: A New Kind of Control for Hybrid Systems* (EP/I001689/1). The first author also acknowledges the support of the Research Councils UK under the grant EP/E50048/1.

References

[1] A. Abate, A. D'Innocenzo, M. Di Benedetto, S. Sastry, Understanding deadlock and livelock behaviors in hybrid control systems, Nonlinear Anal. Hybrid Syst. 3 (2009) 150–162.
[2] B. Akbarpour, L.C. Paulson, MetiTarski: an automatic theorem prover for real-valued special functions, J. Automat. Reason. 44 (3) (2010) 175–205.
[3] B. Alpern, F.B. Schneider, Defining liveness, Inform. Process. Lett. 21 (1985) 181–185.
[4] R. Alur, C. Courcoubetis, T.A. Henzinger, P.-H. Ho, Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems, in: R.L. Grossman, A. Nerode, A.P. Ravn, H. Rischel (Eds.), Hybrid Systems, in: Lecture Notes in Comput. Sci., vol. 736, Springer, 1993.
[5] R. Alur, T. Feder, T.A. Henzinger, The benefits of relaxing punctuality, J. ACM 43 (January 1996) 116–146.

[6] R. Alur, T.A. Henzinger, P.-H. Ho, Automatic symbolic verification of embedded systems, IEEE Trans. Softw. Eng. 22 (3) (1996) 181–201.
[7] M. Aziz-Alaoui, G. Chen, Asymptotic analysis of a new piecewise-linear chaotic system, Internat. J. Bifur. Chaos 121 (1) (2002) 147–157.
[8] E. Baghious, P. Jarry, Lorenz attractor from differential equations with piecewise linear terms, Internat. J. Bifur. Chaos 3 (1) (1993) 201–210.
[9] C. Baier, M. Kwiatkowska, On topological hierarchies of temporal properties, Fund. Inform. 41 (3) (2000) 259–294.
[10] A. Bauer, M. Pister, M. Tautschnig, Tool-support for the analysis of hybrid systems and models, in: Design, Automation and Test in Europe (DATE '07), April 2007, pp. 924–929.
[11] G. Behrmann, K. Larsen, J. Rasmussen, Beyond liveness: efficient parameter synthesis for time bounded liveness, in: P. Pettersson, W. Yi (Eds.), FORMATS 2005, in: Lecture Notes in Comput. Sci., vol. 3829, Springer Berlin/Heidelberg, 2005, pp. 81–94.
[12] M. Benerecetti, M. Faella, S. Minopoli, Revisiting synthesis of switching controllers for linear hybrid systems, in: 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), December 2011, pp. 4753–4758.
[13] M. Benerecetti, M. Faella, S. Minopoli, Automatic synthesis of switching controllers for linear hybrid systems: safety control, Theoret. Comput. Sci. 493 (2013) 116–138.
[14] A. Bouajjani, A. Legay, P. Wolper, Handling liveness properties in (ω-) regular model checking, in: INFINITY 2004, in: Electron. Notes Theor. Comput. Sci., vol. 138, 2005, pp. 101–115.
[15] M. Branicky, Stability of switched and hybrid systems, in: Proceedings of the IEEE Conference on Decision and Control, 1994, pp. 3498–3503.
[16] M. Branicky, Multiple Lyapunov functions and other analysis tools for switched and hybrid systems, IEEE Trans. Automat. Control 43 (4) (1998) 475–482.
[17] R. Carter, E.M. Navarro-López, Abstractions of hybrid systems: formal languages to describe dynamical behaviour, in: 18th IFAC Triennial World Congress, August 31–September 2, 2011, pp. 4552–4557.
[18] R. Carter, E.M. Navarro-López, Dynamically-driven timed automaton abstractions for proving liveness of continuous systems, in: M. Jurdziński, Ničković (Eds.), Formal Modeling and Analysis of Timed Systems, in: Lecture Notes in Comput. Sci., vol. 7595, Springer, 2012, pp. 59–74.
[19] A. Chutinan, B.H. Krogh, Computational techniques for hybrid system verification, IEEE Trans. Automat. Control 48 (2003) 64–75.
[20] E.M. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, M. Theobald, Abstraction and counterexample-guided refinement in model checking of hybrid systems, Internat. J. Found. Comput. Sci. 14 (4) (2003) 583–604.
[21] B. Cook, J. Fisher, E. Krepska, N. Piterman, Proving stabilization of biological systems, in: Proceedings of the Verification, Model Checking, and Abstract Interpretation, VMCAI'11, in: Lecture Notes in Comput. Sci., vol. 6538, Springer-Verlag, 2011, pp. 134–149.
[22] E. Davison, E. Kurak, A computational method for determining quadratic Lyapunov functions for non-linear systems, Automatica 7 (5) (Sept. 1971) 627–636.
[23] R. Decarlo, M.S. Branicky, S. Pettersson, B. Lennartson, Perspectives and results on the stability and stabilizability of hybrid systems, Proc. IEEE 88 (7) (July 2000) 1069–1082.
[24] W. Denman, Automated verification of continuous and hybrid dynamical systems, PhD thesis, University of Cambridge, 2014.
[25] A. Donzé, Breach: a toolbox for verification and parameter synthesis of hybrid systems, in: T. Touili, B. Cook, P. Jackson (Eds.), Computer Aided Verification, in: Lecture Notes in Comput. Sci., vol. 6174, Springer Berlin Heidelberg, 2010, pp. 167–170.
[26] P.S. Duggirala, S. Mitra, Lyapunov abstractions for inevitability of hybrid systems, in: Hybrid Systems: Computation and Control (HSCC), 2012, pp. 115–123.
[27] E. Emerson, J.Y. Halpern, Decision procedures and expressiveness in the temporal logic of branching time, J. Comput. System Sci. 30 (1) (1985) 1–24.
[28] G.E. Fainekos, A. Girard, H. Kress-Gazit, G.J. Pappas, Temporal logic motion planning for dynamic robots, Automatica 45 (2) (2009) 343–352.
[29] H. Flashner, R. Guttalu, A computational approach for studying domains of attraction for non-linear systems, Internat. J. Non-linear Mech. 23 (4) (1988) 279–295.
[30] M. Fränzle, C. Herde, T. Teige, S. Ratschan, T. Schubert, Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure, J. Satisf. Boolean Model. Comput. 1 (2007) 209–236.
[31] R. Ghosh, C.J. Tomlin, An algorithm for reachability computations on hybrid automata models of protein signaling networks, in: 43rd IEEE Conference on Decision and Control (CDC), IEEE, 2004, pp. 2256–2261.
[32] A. Girard, G. Zheng, Verification of safety and liveness properties of metric transition systems, ACM Trans. Emb. Comput. Syst. 11 (S2) (2012) 54.
[33] R. Goebel, R.G. Sanfelice, A.R. Teel, Hybrid dynamical systems: robust stability and control for systems that combine continuous-time and discrete-time dynamics, IEEE Control Syst. Mag. 29 (2009) 28–93.
[34] L. Grujić, On practical stability, Internat. J. Control 17 (4) (1973) 881–887.
[35] J. Guckenheimer, Computing periodic orbits, in: J. Lumley (Ed.), Fluid Mechanics and the Environment: Dynamical Approaches, in: Lecture Notes in Phys., vol. 566, Springer, 2001, pp. 107–119.
[36] H. Guéguen, M.-A. Lefebvre, J. Zaytoon, O. Nasri, Safety verification and reachability analysis for hybrid systems, Ann. Rev. Control 33 (1) (2009) 25–36.
[37] S.J. Hammarling, Numerical solution of the stable, non-negative definite Lyapunov equation, IMA J. Numer. Anal. 2 (3) (1982) 303–323.
[38] M. Hejri, H. Mokhtari, Global hybrid modelling and control of a buck converter: a novel concept, Internat. J. Circuit Theory Appl. 37 (2008) 968–986.
[39] T.A. Henzinger, P.W. Kopke, A. Puri, P. Varaiya, What's decidable about hybrid automata?, J. Comput. System Sci. 57 (1998) 94–124.
[40] J. Hopcroft, R. Motwani, J. Ullman, Introduction to Automata Theory, Languages, and Computation, 3rd edition, Addison–Wesley, 2007.
[41] A.H. iSAT Developer Team, iSAT quick start guide, 2010, available online at http://isat.gforge.avacs.org (accessed on 26th July 2012).
[42] K.H. Johansson, M. Egerstedt, J. Lygeros, S. Sastry, On the regularization of Zeno hybrid automata, Systems Control Lett. 38 (3) (October 1999) 141–150.
[43] H. Kowshik, D. Caveney, P. Kumar, Safety and liveness in intelligent intersections, in: HSCC 2008, in: Lecture Notes in Comput. Sci., vol. 4981, 2008, pp. 301–315.
[44] R. Koymans, Specifying real-time properties with metric temporal logic, Real-Time Syst. 2 (1990) 255–299.
[45] V. Lakshmikantham, S. Leela, A. Martynyuk, Practical Stability of Nonlinear Systems, World Scientific Publishing, 1990.
[46] D. Liberzon, Switching in Systems and Control, Birkhäuser, 2003.
[47] J. Liu, J. Lu, X. Wu, Bridge the gap between the PWL Lorenz and PWL Chen's system, in: Proceedings of the 8th International Conference on Control, Automation Robotics and Vision, vol. 1, 2004, pp. 1368–1373.
[48] J. Lygeros, K.H. Johansson, S.N. Simic, S.S. Sastry, Dynamical properties of hybrid automata, IEEE Trans. Automat. Control 48 (1) (January 2003) 2–17.
[49] C. Mitrohin, A. Podelski, Composing stability proofs for hybrid systems, in: U. Fahrenberg, S. Tripakis (Eds.), FORMATS, in: Lecture Notes in Comput. Sci., vol. 6919, Springer Berlin/Heidelberg, 2011, pp. 286–300.
[50] E.M. Navarro-López, J.G. Barajas-Ramírez, Bringing order to chaos: hybrid modelling of a discontinuous chaotic system, in: Proceedings of the 11th International Workshop on Variable Structure Systems, 2010, pp. 325–330.
[51] E.M. Navarro-López, R. Carter, Hybrid automata: an insight into the discrete abstraction of discontinuous systems, in: Special Issue on Variable Structure Systems Methods for Control and Observation of Hybrid Systems, Internat. J. Systems Sci. 42 (11) (2011) 1883–1898.
[52] E.M. Navarro-López, D. Cortés, Avoiding harmful oscillations in a drillstring through dynamical analysis, J. Sound Vib. 307 (2007) 152–171.
[53] E.M. Navarro-López, D. Cortés, C. Castro, Design of practical sliding-mode controllers with constant switching frequency for power converters, Electric Power Syst. Res. 79 (5) (2009) 796–802.
[54] E.M. Navarro-López, D. Laila, Group and total dissipativity and stability of multi-equilibria hybrid automata, IEEE Trans. Automat. Control 58 (12) (December 2013) 3196–3202.

[55] Y. Ohta, H. Imanishi, L. Gong, H. Haneda, Lyapunov functions for a class of nonlinear systems, IEEE Trans. Circuits Syst. I Fund. Theory Appl. 40 (5) (May 1993) 343–354.
[56] S. Owicki, L. Lamport, Proving liveness properties of concurrent programs, ACM Trans. Program. Lang. Syst. 4 (3) (July 1982) 455–495.
[57] M. Paul, H.J. Siegert, M.W. Alford, J.P. Ansart, G. Hommel, L. Lamport, B. Liskov, G.P. Mullery, F.B. Schneider (Eds.), Distributed Systems: Methods and Tools for Specification. An Advanced Course, Lecture Notes in Comput. Sci., vol. 190, Springer Berlin/Heidelberg, 1985.
[58] A. Platzer, J.-D. Quesel, KeYmaera: a hybrid theorem prover for hybrid systems (system description), in: A. Armando, P. Baumgartner, G. Dowek (Eds.), Automated Reasoning, in: Lecture Notes in Comput. Sci., vol. 5195, Springer Berlin/Heidelberg, 2008, pp. 171–178.
[59] A. Pnueli, The temporal logic of programs, in: Proceedings of the 18th International Symposium on the Foundations of Computer Science, 1977, pp. 46–57.
[60] A. Podelski, S. Wagner, Model checking of hybrid systems: from reachability towards stability, in: Hybrid Systems: Computation and Control, in: Lecture Notes in Comput. Sci., vol. 3927, 2006, pp. 507–521.
[61] A. Podelski, S. Wagner, A sound and complete proof rule for region stability of hybrid systems, in: Hybrid Systems: Computation and Control, in: Lecture Notes in Comput. Sci., vol. 4416, 2007, pp. 750–753.
[62] S. Prajna, A. Jadbabaie, Safety verification of hybrid systems using barrier certificates, in: R. Alur, G. Pappas (Eds.), Hybrid Systems: Computation and Control, in: Lecture Notes in Comput. Sci., vol. 2993, Springer, 2004, pp. 477–492.
[63] S. Ratschan, Z. She, Safety verification of hybrid systems by constraint propagation-based abstraction refinement, ACM Trans. Emb. Comput. Syst. 6 (1) (2007) 573–589.
[64] S. Ratschan, J.-G. Smaus, Verification-integrated falsification of non-deterministic hybrid systems, in: Proceedings of the 2nd IFAC Conference on Analysis and Design of Hybrid Systems, 2006, p. 371.
[65] B. Silva, O. Stursberg, B. Krogh, S. Engell, An assessment of the current status of algorithmic approaches to the verification of hybrid systems, in: Proceedings of the 40th IEEE Conference on Decision and Control (CDC), vol. 3, 2001, pp. 2867–2874.
[66] C. Sloth, R. Wisniewski, Verification of continuous dynamical systems by timed automata, Form. Methods Syst. Des. 39 (1) (August 2011) 47–82.
[67] J.A. Stiver, X.D. Koutsoukos, P.J. Antsaklis, An invariant-based approach to the design of hybrid control systems, Internat. J. Robust Nonlinear Control 11 (April 2001) 453–478.
[68] O. Stursberg, S. Kowalewski, I. Hoffmann, J. Preussig, Comparing timed and hybrid automata as approximations of continuous systems, in: Hybrid Systems IV, Springer, 1997, pp. 361–377.
[69] K.-C. Tai, Definitions and detection of deadlock, livelock, and starvation in concurrent programs, in: International Conference on Parallel Processing, vol. 2, 1994, pp. 69–72.
[70] C. Tomlin, G.J. Pappas, S. Sastry, Conflict resolution for air traffic management: a study in multiagent hybrid systems, IEEE Trans. Automat. Control 43 (4) (April 1998) 509–521.
[71] V. Utkin, Sliding Modes in Control Optimization, Springer-Verlag, 1992.
[72] L. Vandenberghe, S. Boyd, A polynomial-time algorithm for determining quadratic Lyapunov functions for nonlinear systems, in: Proceedings of the European Conference on Circuit Theory and Design, 1993, pp. 1065–1068.
[73] L. Weiss, E. Infante, On the stability of systems defined over a finite time interval, Proc. Natl. Acad. Sci. USA 54 (1) (1965) 44–48.
[74] L. Weiss, E. Infante, Finite time stability under perturbing forces and on product spaces, IEEE Trans. Automat. Control 12 (1) (1967) 54–59.